НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____ Сергій, СТІРЕНКО

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Інформаційна платформа для пошуку подій за інтересами (серверна частина)»

Виконав (-ла):
студент (-ка) IV курсу, групи ІП-64
Вінницький В'ячеслав Андрійович _____

Керівник:
Асистент кафедри ОТ,
Каплунов Артем Володимирович _____

Консультант з нормконтролю:
Професор, доктор технічних наук
Сімоненко Валерій Павлович _____

Рецензент:

_____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт.	2	
2	А4	ДП 6402. 00.001 ВП	Відомість дипломного проєкту.	1	
3	А4	ДП 6402. 00.002 ТЗ	Технічне завдання	3	
4	А4	ДП 6402. 00.003 ПЗ	Пояснювальна записка	60	
5	А4	ДП 6402. 00.004 Д1	Функціональна схема.	1	
6	А4	ДП 6402. 00.005 Д2	Схема бази даних.	1	
7	А4	ДП 6402. 00.006 Д3	Схема пакетів системи.	1	
8	А4	ДП 6402. 00.007 Д4	Принципова схема схема.	1	

				ДП 6402 00.001		
	ПІБ	Підп.	Дата			
Розробн.	Вінницький В.А.			Відомість дипломного проєкту	Лист	Листів
Керівн.	Каплунов А.В.				1	1
Консульт.					КПІ ім. Ігоря Сікорського Каф. ОТ Гр. ІП-64	
Н/контр.	Сімоненко В.П.					
Зав.каф.	Стіренко С.Г.					

Національний технічний університет України

«Київський політехнічний інститут»

Інститут (факультет) інформатики та обчислювальної техніки

(повна назва)

Кафедра обчислювальної техніки

(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 121 «Інженерія програмного забезпечення»

(повна назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____Сергій, СТІРЕНКО
"___" _____ 2020 року

ЗАВДАННЯ

на дипломний проєкт студента

_____В’ячеслав ВІННИЦЬКИЙ_____
(ім'я, прізвище)

1. Тема проєкту *«Інформаційна платформа для пошуку подій за інтересами (серверна частина)»*

керівник проєкту *асистент кафедри ОТ, Артем КАПЛУНОВ* _____,
(ім'я, прізвище, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "___" _____ 20__ року N ___

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту __технічна документація, теоретичні дані, інтернет-публікації за темою роботи_____
_____
_____
_____

4. Зміст пояснювальної записки

− Провести огляд і порівняння існуючих рішень
− Виконати дослідження з доступних засобів для розробки веб-сервісу
− Виконати розробку системи з пошуку подій
− Провести моделювання та аналіз розробленого методу

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)
_____
_____
_____

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	*асистент кафедри Каплунов А.В.*	*10.03.2020*	*31.05.2020*
2	*асистент кафедри Каплунов А.В.*	*10.03.2020*	*31.05.2020*
3	*асистент кафедри Каплунов А.В.*	*10.03.2020*	*31.05.2020*
4	*асистент кафедри Каплунов А.В..*	*10.03.2020*	*31.05.2020*

7. Дата видачі завдання *15.12.2019*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів проекту	Примітка
1	*Затвердження теми роботи*	*15.12.2019*	
2	*Вивчення та аналіз завдання*	*20.12.2019-02.02.2020*	
3	*Проведення огляду алгоритмів та підходів для розробки системи*	*02.02.2020-01.03.2020*	
4	*Проведення аналізу засобів для розробки програмного забезпечення*	*01.03.2020-15.04.2020*	
5	*Розробка програмної реалізації веб-сервісу*	*15.04.2020-13.05.2020*	
6	*Проведення тестування та аналізу розробленої системи*	*13.05.2020-19.05.2020*	
7	*Оформлення матеріалів роботи*	*19.05.2020-31.05.2020*	
8	*Передзахист*		
9	*Захист*		

Студент _____ Вінницький В.А.
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ Каплунов А. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

В бакалаврській дипломній роботі було запропоновано та реалізовано програмний продукт формату соціальної мережі для пошуку подій за інтересами. Програма дозволяє дивитися актуальні події у своїй місцевості, а також створювати свої власні події, які будуть видні іншим користувачам. Таким чином платформа допомагатиме людям у соціалізації на основі схожих інтересів, знаходити нові знайомста на цікавих заходах та проводити час приємно та корисно. Також платформа повинна допомгти людям знайти однодумців або ж розширити власний кругозір за допомгою відвідування неформальних заходів. В порівнянні з існуючими продуктами ця платформа є безбечною, вільною від обробки персональних даних, позбавлена рекламних пропозицій. Програмний продукт було створено на мові Java.

ABSTRACT

In the bachelor's thesis, a software product in the format of a social network for searching for events by interests was proposed and implemented. The program allows you to watch current events in your area, as well as create your own events that will be visible to other users. In this way, the platform will help people to socialize based on similar interests, find new acquaintances at interesting events and spend time pleasantly and usefully. The platform should also help people find like-minded people or broaden their horizons by attending informal events. Compared to existing products, this platform is safe, free from personal data processing, free of advertising offers. The software product was created in Java.

Технічне завдання до дипломного проекту

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА СПОСІБ ВИКОРИСТАННЯ

Дане технічне завдання розповсюджується на розробку платформи для пошуку подій за інтересами користувача.

Спосіб використання: перегляд та відвідування актуальних подій у своїй місцевості, а також створення своїх власних заходів, які будуть переглядатися іншими користувачами платформи.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання розробки серверної частини інформаційної платформи для пошуку події, затвердженою кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка серверного додатку для платформи пошуку подій за інтересами.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами інформаціїї для розробки є відкриті джерела з простору Інтернету, науково-технічні статті на пересічні теми та профільна технічна література.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- Розробка веб-сервісу для збереження, обробки та передачі данних для платформи пошуку подій.

5.2. Вимоги до програмного забезпечення для серверного програмного забезпечення:

- Операційна система MacOS Yosemite або новіше, MS Windows XP, MS Windows 7, Linux Mint, MS Windows 8/8.1, MS Windows 10, Linux Ubuntu 14.04 або новіше.
- JRE 11 або новіше.

5.3. Вимоги до апаратного забезпечення

- Комп’ютер з процесором Intel Core 2 Duo або новіше.
- Оперативна пам’ять обсягом не менше 512 Мбайт.
- Вільне місце на дисковому просторі не менше 500 Мбайт.
- Підключення до мережі Інтернет.

Пояснювальна записка
до дипломного проєкту
на тему: «Інформаційна платформа для пошуку подій за інтересами (серверна частина)»

Київ – 2020 року

ЗМІСТ

ВСТУП

На даний момент люди проводять в мережі інтернет близько третини свого вільного часу і ця цифра росте з кожним роком. Проникнення різноманітних сервісів, платформ, соцальних мереж у життя кожного з нас стає все більше й більше з кожним днем. Велика кількість багатомільярдних компаній намагається створити цифрове майбутнє планети, постійно вдосконалюючи свої продукти. Майже всі сфери реального життя мають якийсь певний аналог чи помічник у вигляді програмного продукту. Нк є винятком і сфера подій, зустрічей, які люди знаходять онлайн, а потім відвідують особисто. Такі програмні продукти бувають різні : деякі під це спеціалізовані, інші є маленькою частиною величезної екосистеми, яка і формує перелік доступних у додатку подій та зустрічей.

Актуальність теми дипломної роботи полягає в необхідності розробки і створення серверної частини для платформи пошуку подій за інтересами, яка буде надавати доступ до даних та функціоналу платформи для клієнтських додатків.

Практичним значенням побудованої системи є надання користувачам платформи можливості бути в центрі подій : відвідувати заходи, ділитися подіями з друзями, створювати свої власні, що може бути дуже корисним для активних, зацікавлених у нових знайомствах та яскравих емоціях.

Об'єктом даного дослідження є методи розробки, тестування та розгортання на серверах даного рішення.

Метою роботи є збір та систематизація знань про розробку складних систем на основі мобільного додатку та хмарного сховища та створення реалізації для вирішення вирішення поставленої задачі.

Задачі, що поставлення для досягнення мети даної роботи:

- Розглянути та порівняти основні сервіси, що застосовуються для пошуку подій, проаналізувавши їх переваги та недоліки між собою.
- Дослідити достіпні засоби для розробки веб-сервісу, алгоритмічних складових для нього та баз даних.

- Створити серверну частину платформи пошуку подій за інтересами.
- Проаналізувати показники швикості та безпеки створеної системи та знайти шляхи для оптимізації наданого рішення.

РОЗДІЛ 1
ОГЛЯД І ПОРІВНЯННЯ ІСНУЮЧИХ РІШЕНЬ

1.1. Актуальність теми дослідження

Проект - це клієнт-серверна платформа для пошуку подій за інтересами. Користувачі можуть дивитися актуальні події у своїй місцевості, а також створювати свої власні події, які будуть видні іншим користувачам.

Зайшовши на івент - користувач побачить назву івенту, його опис, творця, місце розташування, час початку події, список користувачів, які йдуть на подію, а також можливість підтвер свою присутність на подію. У користувача буде можливість шукати події на карті щоб планувати свою поїздку на зацікавленість подія (наприклад в іншому місті). Також буде можливість шукати заходи в пошуку (в тому числі, які вже пройшли) або зайти знайти іншого користувача / друга в додатку. За допомогою цього додатка люди можуть знаходити однодумців, допомагати один одному, бути в курсі цікавих подій починаючи від домашніх гуртків по Arduino - закінчуючи величезними концертами рок виконавців, а також проводити час весело разом.

1.2. Поняття соціальної мережі

Соціальна мережа - це платформа для спілкування, обговорення, новин, місце, де люди можуть поділитися новинами зі свого життя, зустріти знайомих, завести нових друзів. У сучасному світі соціальні мережі стали відправною точкою інтернету, тобто більшість людей заходячи в інтернет шукають саме соціальну мережу. Першою і найуспішнішою соціальною мережею є Facebook, ця платформа налічує вже понад півтора мільярда користувачів. Це майже половина користувачів усієї всесвітньої павутини. Тобто з впевненістю можна сказати, що соціальні мережі - найбільш популярне місце в інтернеті. Крім Facebook існує безліч інших соціальних мереж, напимер "Вконтакте" і "Однокласники" в Росії, Weibo в Китаї, Twitter, Youtube, Tinder, LinkedIn. Так як конкурувати з Facebook складно, то інші соціальні мережі намагаються фокусуватися на певному ринку як Weibo, "Вконтакте", "Однокласники" або аудиторії як Tinder або LinkedIn.

Сучасні соціальні мережі це більше ніж просто сайт, це цілий світ. Блукаючи по Фейсбуку людина може знайти смішну картинку, потім написати під нею коментар, а пізніше познайомитися з незнайомою людиною. Багато людей ведуть свій бізнес використовуючи соціальну мережу як платформу. Наприклад якщо дрібний бізнес хоче створити собі сайт для залучення аудиторії до свого провадження то їм набагато простіше створити безкоштовну сторінку на Twitter і групу на Facebook популяризуючи свої послуги. Їм не потрібно піклуватися кількості людей на платформі так як про це вже подбали розробники соціальної мережі. Тобто з впевненістю можна сказати що соціальні мережі намагаються скопіювати і перенести в режим онлайн всі соціальні ритуали які ми робимо в реальному житті.

1.3. Огляд існуючих сервісів

Під час роботи над дипломним проектом було прийнято рішення аналізу відомих технічних рішень, що існують на ринку та розглядаються як потенційні конкуренти. Даний аналіз дасть змогу оцінити різні підходи до створення вузькоспеціальзованих соціальних мереж, проаналізувати недоліки та переваги кожного з них.

1.3.1. Facebook Local

Facebook Local - це порівняно новий сервіс, який останнім часом набуває все більшої популярності. Це тому, що Local - це насправді події, що створювалися на Facebook, але у вигляді окремого додатку.

Найновіша програма Facebook про події доступна для iOS та Android. Як ми можна прочитати у розділі "про" магазину, головною задачою додатку є інформування користувачів про подіїї, що відбуваються неподалік, чи створення можливсті зайти як гарно провести вечір із друзями або ж дослідити околиці. Facebook Local - це сучасніша, вдосконалена версія програми Facebook Events, яка завоювала серця користувачів майже рік тому. Оновлений локальний надає нові можливості, оскільки поєднує події та місця

в одній пошуковій системі. Він збирає дані з 70 мільйонів бізнес-сторінок Facebook, а також огляди та реєстрації друзів. По суті, це означає виявити, що відбувається поруч. Він поєднує інформацію про події, місцевий бізнес та місця. Користувачеві більше не потрібно копатись у головному додатку Facebook серед хаосу новин, щоб знайти події, які їх цікавлять. Користувачі порівнюють його з такими додатками, як Foursquare та Yelp, коли він здійснює пошук та виявлення на локальному рівні. Потім представляє результати в одному місці у вигляді списку. Простий і зрозумілий інтерфейс, а також налаштування в кожній області полегшує доступ до будь-якої події, сторінки або місця, що відображається перед користувачем. Прямо вгорі домашньої сторінки користувачів програми "Місцеві" можна побачити сьогоднішню дату та місцеву погоду. Коли вони трохи поглянуть вниз, є безліч варіантів, які допоможуть їм знайти найкращі місця та заходи у своєму місті. Це дуже зручно, коли ви відвідуєте інше місто (або країну) і хочете проводити вільний час через години. Або коли ви плануєте відрядження і хочете взяти участь у деяких місцевих ділових (або культурних) ініціативах. Наступний розділ у додатку Facebook Local - це путівники. Вибір подій та місць тут широкий. Від їжі, нічного життя, музики та фітнесу до мистецтва та мереж. Усі події можуть бути представлені у вигляді точок на карті розташування користувачів. Це чудово, оскільки допомагає їм уявити, куди їхати, як далеко та які привабливі напрямки поблизу. Він організовує заходи в хронологічному порядку та заохочує користувачів скласти власний графік. Якщо користувач зацікавлений у події, він може натиснути кнопку, зберегти її календар і - так само, як це працює з додатком Facebook - їм надаються сповіщення та інформація про подію. Це також про місця та місця призначення. Події є основною частиною Facebook Local, але додаток також є посібником, який може отримати своїх користувачів через місцеві атракціони, ресторани, кафе чи бари. У додатку перелічені місця поблизу, а також їхні рейтинги та інформація про друзів користувача у Facebook, які раніше там були зареєстровані. Це незвично,

оскільки тепер користувачі вибирають місце, де рекомендують друга, а не відгук незнайомця.

Перелік можливостей, які продукт надає користувачам:

- Перегляд останніх дій, заходів і місць, з якими взаємодіють ваші друзі, а також перегляд новин від організаторів заходів і Сторінок, на які ви підписані.
- Пошук заходів і дій поруч з вами на інтерактивній карті. Можливість фільтрації за часом, категорії, місця і так далі.
- Отримання рекомендацій з урахуванням того, що популярно серед ваших друзів, чим ви займалися в минулому і які Сторінки вам подобаються. Перегляд гідів по різних місцях.
- Можливість додавання календарів з телефону, щоб переглядати всі плани в одному місці.
- Можливість підписатися на оновлення про найближчі заходи, щоб завжди бути в курсі всіх змін.

1.3.2. MeetUp

Meetup - це додаток для пошуку та створення нових локальних взаємозв'язків. Ви можете буди користувачем платформи для проведення зустрічей, знайомства з новими людьми, пошуку підтримки у якомусь своєму проекті, навчанні, реалізаціїї своїх задумок або просто виходу зі своєї зони комфорту. І платформа дає можливість робити це неподалік від дому. За допомого визначення геолокації через GPS-датчики у пристроях рекомендуються зустрічі, що проходять поблизу.

Перелік можливостей, які продукт надає користувачам:

- Знаходити місцеві заходи та групи: збори книжкових клубів, безкоштовні уроки йоги і багато іншого.
- Вибирайте цікаві категорії, шукайте за ключовими словами або дізнавайтеся, що популярно в вашому регіоні.
- Зберігати цікаві події і стежити за ними.

- Спілкуйтеся з людьми, яких ви зустріли на заході, надсилайти їм повідомлення і обговорювати ідеї.
- Змінити місто чи країну, щоб дізнатися, які заходи проводяться там.
- Створити групу для людей зацікавлених одним проектом.
- Стати організатори заходу, щоб знайомитися з людьми, з якими у вас спільні інтереси.
- Призначати заходи і керувати групою з будь-якої точки Землі з доступом до Інтернету.

1.3.3. NextDoor

Nextdoor – це додаток, що концентрується на подіях вашого мікрорайону. Він надає можливість знайти заходи поруч, запропонувати якісь свої речі на продаж, або знайти чудові речі, що доступні безкоштовно у вашому районі. Також використовувати додаток можна, щоб бути в курсі того, що відбувається у вашому районі, наприклад зв'язатися зі своїми сусідами, знайти помічника для домашніх справ, або просто краще дізнатися людей, яких ви зустрічаєте щодня. Знайдіть розпродаж у дворі, майстра-професіонала, вигулу собак або няні. Nextdoor дозволяє легко говорити з сусідами про те, що для вас найбільш важливо. Користувач має змогу спілкуватися зі своїми сусідами для того, щоб бути усвідомленим про місцеві новини, планувати заходи або давати сусідам порадами. Платформа має широкий спект видів діяльнсті - від громадської до продажу дворів. З'єднуючи сусідів, додаток підтверджує, що вони можуть нам допомогти у багатьох аспектах – корисувачі просто повинні з ними зв’язатися.

Перелік можливостей, які продукт надає користувачам:

- Отримувати нагадування про заходи, що відбуваються в околицях
- Отримувати рекомендації щодо деяких послуг від сусідів
- Розмістити оголошення, щоб запропонувати сусідам послугу
- Отримати безкоштовно певні речі, коли сусід влаштовує ярмарку своїх підтриманих речей

- Знайти маленькі місцеві події, наприклад кулінарія чи громадські дії
- Організуйте околиці, щоб реагувати на злочині дії
- Ділитися інформацією під час стихійного лиха

1.3.4 Tinder

Tinder - це програма для знайомств, яка дозволяє людям анонімно проводити пальцем ліворуч або праворуч на когось на основі їх картини, невеликої біографії та загальних інтересів. Революційний (на той час) Tinder "проведіть пальцем праворуч, якщо вона вам подобається, проведіть пальцем ліворуч, якщо ні", з тих пір скопійовано численними конкурентами, і таких програм, як Tinder, є багато.

Перелік можливостей, які продукт надає користувачам:

- Пошук: це можливість для пошуку нових людей
- Місцезнаходження: вказати поточне місцеположення
- Відстань: можливість вказати інтервал, в межах якого ви хочете зустрітися з людьми.
- Віковий інтервал: можливість обрати вік, за яким буде відбуватися пошук людей. Вас також побачать люди, які визначили діапазон віку, що підпає під ваш.

1.4. Порівняння існуючих сервісів

Проаналізувавши існуючі сервіси на прикладі наведених вище додатків можна виділити, що всі з них пропронують людині соціалізацію за інтересами. Наприклад, Facebook Local фокусується на пошуку груп людей які вже об’єднані за певними критеріями. MeetUp дозволяє відвідувати лекції, семінари та подіїї переважно за професійними інтересами. NextDoor претендує бути платформою, яка є збирає в собі всі активності що відбуваються у вашому районі, тобто об’єднує людей за географічною ознакою. Tinder дозволяє зайти окремих людей за ознакою віку, статі та відстанню. З цього можна зробити висновок, що соціальний додаток обов’язково повинен містити функцію

розширення кола знайомих за певними спільними ознаками. Тому є сенс зробити у веб-сервісі акцент на підборі релевантних подій для користувача.

ВИСНОВКИ ДО РОЗДІЛУ 1

Дослідивши актуальність теми дослідження стає зрозуміло, що соціальні мережі та взаємодія людей в Інтернеті стають все більш і більш звичними з кожним днем. На фоні цього створюється велика кількість додатків, що покликані для спрощення та покращення життя. Проаналізувавши сферу соціальних мереж, що орієнтовані на подіїї, можна сказати, що обов'язково у сучасному соціальному додатку повинна бути можливість розширення кола знайомих. Таким чином люди зможуть не лише знаходити у додатку поодії, а і наприклад знайомитися з іншими людьми на профільних заходах. Також дуже важливою частиною для успіху додатку є позиціонування, вибір аудиторії на яку націлений сервіс. Найбільш переспективною є аудиторія молодих людей 18-35 років, що ведуть активний образ життя та обожнюють знаходити нові знайомства або відвідувати заходи на будь-який смак. Таким чином, будучи користувачами вони будуть не тільки переглядати події, а й відвідувати їх та створювати свої власні і запрошувати на них друзів. Іншою характерною особливістю існуючих сервісів є те, що вони мають різні способи монетизації, тому на даний момент можна прийняти, що додаток буде без монетизаціїї

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ДОСТУПНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ ВЕБ-СЕРВІСУ

2.1. Аналіз та порівняння існуючих алгоритмів

Варто розглянути дослідження дослідження та публікацій на рахунок алгоритмів, технологій та кращих практик для реалізації опорних частнин веб-сервісу. Опорними частинами слід визначити визначення релевантних подій для конкретного користувача та структуризація подій за їх локацією.

2.1.1. Огляд існуючих алгоритмів

Основною сторінкою, яку найчастіше переглядає користувач є стрічка подій. Для того, щоб збільшити кількість часу яку контрентний користувач буде проводити у додатку основна сторінка повинна бути персоналізована та відображена згідно його побажать. Це перша область у якій планується бути задіяні певні алгоритмічні застосування. Другим за важливістю завданням є вибірка подій за геолокацією, отже також варто зосередитися на алгоритмах, що можуть бути корисними за таких умов.

2.1.1.1. Кластеризація

Методи кластеризації [1] - вагома частина алгоритмів машинного навчання. Вони можуть допомогти створити певну модель за якої кожна подія буде занесена до певної групи. Маючи інформацію про те, які саме групи інтересів у користувача може бути спрощено процес відображення користувачу подій, що будуть йому цікавими.

2.1.1.1.1. Метод k-середніх

Метод k-середніх [2] – один із найпопулярніних методів кластеризації - упорядкування деякої кількості об'єктів у відносно однорідні групи за певними ознаками. У 1950-х роках його майже одночасно, але працюючи одноосібно, винайшли математики Уго Штайнгауз і Стюарт Ллойд. Ціль методу - розділити k спостережень на n кластерів, таким чином, що кожне

спостереження буде належати кластеру з найближчим до нього середнім значенням. Метод заснований на зменшеннні суми квадратів відстаней між центром кластера та кожним спостереженням, тобто функції $\sum_{i=1}^{N} d(x_i, m_j(x_i))^2$, де

d — метрика,

x_i — і-ий об'єкт даних,

$m_j(x_i)$ — центр кластера, якому на j-ій ітерації рівний елемент x_i.

Як вхідні дані є масив певних об'єктів, кожен з яких має характеристики з різними значеннями. В залежності від своїх характеристик об'єкти розташовується у багатовимірному просторі.

1. Визначається кількість кластерів, яку необхідно утворити
2. Випадковим чином обирається k об’єктів, які будуть вважаютися центрами кластерів на цьому кроці
3. Кожен об’єкт присвоюється до одного з n кластерів відстань до якого найкоротша
4. Новий центр для кожного кластера розраховується як елемент, ознаки якого розраховуються як середнє арифметичне значення характеристик об'єктів, що входять у цей кластер
5. Кроки 3-4 повторюються до того моменту, поки кластерні центри стануть стійкими (тобто при кожній ітерації в кожному кластері знаходитимуться одні й ті самі об'єкти), таким чином дисперсія всередині кластера буде мінімізована, а між кластерами — максимізована

Вибір кількості кластерів відбувається на основі дослідницької гіпотези. Якщо її немає, то рекомендують створити 2 кластери, далі 3, 4, 5, порівнюючи отримані результати.

На початку було випадково згенеровано деяку кількість точок та серед них обрано k початкових «середніх» (тут k=3) об’єктів у межах домена даних (кольорові).

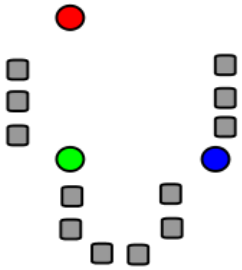


Рис 2.1. Крок 1.

Створено k кластерів, асоціюючи кожне спостереження з найближчим середнім. Розбиття відбувається згідно з діаграмою Вороного утвореною середніми. Тобто береться метричний простір який заповнюється.

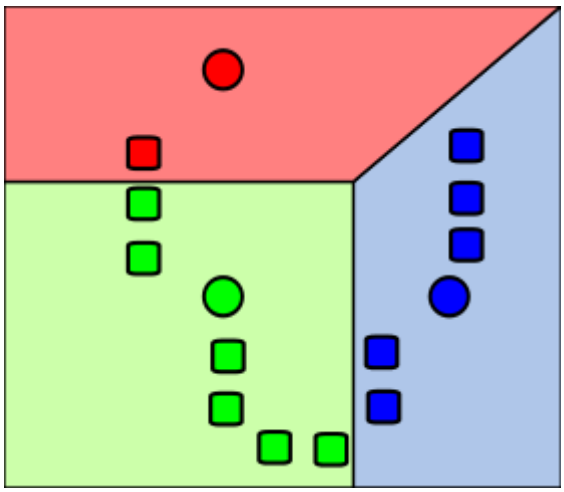


Рис 2.2. Крок 2.

Після чого центроїд, тобто елемент, ознаки якого розраховуються як середнє арифметичне значення характеристик об'єктів, що входять у цей кластер кожного з k кластерів стає новим середнім. Цей крок дозволяє підсунути елементи або ближче або далі від цільового кластеру.

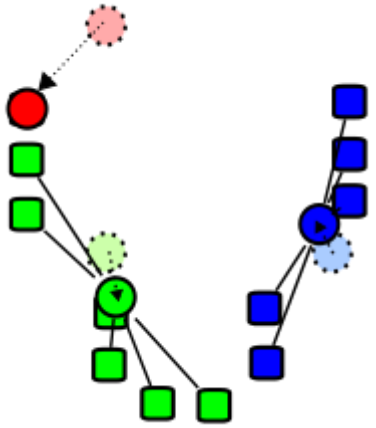


Рис 2.3. Крок 3

Кроки 2 і 3 ітеруються поки не буде досягнута збіжність, тобто до моменту коли кожен кластер міститиме в собі однакові елементи в незалежності від ітерації. Принцип алгоритму полягає в пошуку таких центрів кластерів та наборів елементів кожного кластера при наявності деякої функції Ф(°), що виражає якість поточного розбиття множини на k кластерів, коли сумарне квадратичне відхилення елементів кластерів від центрів цих кластерів буде найменшим:

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2$$, де

k — число кластерів,

S_i — отримані кластери, $i = 1, 2, \ldots, k$, μ_i — центри мас векторів.

В початковий момент роботи алгоритму довільним чином обираються центри кластерів, далі для кожного елемента множини ітеративно обраховується відстань від центрів з приєднанням кожного елемента до кластера з найближчим центром. Для кожного з отриманих кластерів обчислюються нові значення центрів, намагаючись при цьому мінімізувати функцію Ф(°), після чого повторюється процедура перерозподілу елементів між кластерами. Алгоритм методу «Кластеризація за схемою к-середніх»:

- вибрати k інформаційних точок як центри кластерів поки не завершиться процес зміни центрів кластерів;

- зіставити кожну інформаційну точку з кластером, відстань до центра якого мінімальна;
- переконатися, що в кожному кластері міститься хоча б одна точка. Для цього кожний порожній кластер потрібно доповнити довільною точкою, що розташована «далеко» від центра кластера;
- центр кожного кластера замінити середнім від елементів кластера;
- кінець.

2.1.1.2. Ранжування

Формальна постановка задачі ранжування виглядає наступним чином. Є вибірка, що складається з *l* елементів. Вона складається з об'єктів, що мають опис відмінності, як і в завданнях регресії і класифікації. Ключова відмінність ранжування полягає в цільовій змінній. Якщо в задачах навчання з учителем кожному об'єкту відповідає своя відповідь, то тут відповіді - це пари виду:

$\{(l, j): x_i < x_j\}$.

Набір таких пар і є цільовою змінною.

В задачі потрібно побудувати ранжуючу модель a(x) таку, що:

$x_i < x_j \Rightarrow a(x_i) < a(x_j)$.

Таким чином, по виходах необхідно відновити порядок, а величина відповідей не має значення. В цьому полягає головна відмінність ранжирування від інших завдань машинного навчання. Основне застосування ранжування - це ранжування пошукової видачі. Наприклад, в Яндексі в якості об'єктів виступають пари запит, документ [3]. Запит - це ключові слова, введені користувачем, документ - один з усіх документів, наявних в пошуковому індексі: (Запит, документ1) < (Запит, документ2).

Завдання - навчитися відновлювати порядок на таких парах, причому запит у всіх парах один і той же. Таким чином, потрібно ранжувати документи для одного запиту. Для навчання такий порядок задається асессорами. Це

спеціальні люди, які отримують пари запит, документ і оцінюють релевантність документа, те, наскільки добре він відповідає даному запиту.

Оцінки можуть бути як числовими, так і попарними (асесор в парі документів вибирає той, який більш релевантним запитом). Інший приклад застосування ранжування - це рекомендації. У цьому завданні по парам користувач, товар потрібно ранжувати товари для даного користувача, максимізуючи при цьому деяку метрику. Порядок задається на підставі уподобань користувача, його попередніх покупок і оцінок.

2.1.1.2.1. Особливості задачі

У завдання ранжування є свої особливості. По-перше, об'єкти не є незалежними: цільова змінна залежить від пар об'єктів. Це необхідно враховувати при вирішенні. По-друге, використовуються складніші метрики, ніж для регресії або класифікації. Зазвичай вони дискретні, тому що важливі не отримують значення, а порядок, який задає модель. Ще одна особливість - для цього завдання дуже важливо правильно сформувати вибірку, тому що це можна зробити безліччю способів. Не зрозуміло, наприклад, які саме пари

віддавати на розмітку асесор в завданні ранжування пошукової видачі, або як саме отримувати переваги користувачів з даних для завдання побудови рекомендацій. Для простоти всюди далі в якості прикладу візьмемо завдання ранжування пошукової видачі, але все міркування добре узагальнюються і для інших застосувань. У найпростішій постановці завдання ранжирування цільова змінна приймає два значення, документ або релевантним запитом, чи ні: $y(q, d) \in \{0, 1\}$, де y - цільова змінна, q - запит, d - документ.

Нехай також є деяка модель a(q, d), яка оцінює релевантність документа запиту. За значеннями, отриманими за допомогою цієї моделі, можна віжранжувати документи. $d^{(i)}_q$ буде позначати і-й по релевантності документ для запиту q. Після того, як введені позначення, можна задати найпростішу метрику ранжирування. Це Precision@k, точність серед перших k документів (k - параметр метрики). Якщо ранжируется пошукова видача, і на першій

сторінці показуються 10 документів, то розумно вибирати k = 10. Дана метрика визначається як частка релевантних документів серед перших k, отриманих за допомогою моделі:

$$\text{Precision@}k(q) = \frac{1}{k}\sum_{i=1}^{k} y\left(q, d_q^{(i)}\right),$$

Це корисна метрика, тому що зазвичай важливо мати релевантні документи серед перших k документів. Однак у неї є серйозний недолік: позиції релевантних документів ніяк не враховуються. Наприклад, якщо при k = 10 серед перших k документів є 5 релевантних, то не важливо, де вони знаходяться: серед перших або останніх 5 документів. Зазвичай же хочеться, щоб релевантні документи розташовувалися якомога вище.

Описану проблему можна вирішити, модифікувавши метрику, і визначити середню точність.

Дана величина вже залежить від порядку. Вона досягає максимуму, якщо все релевантні документи знаходяться вгорі ранжированного списку. Якщо вони зміщуються нижче, значення метрики зменшується.

І точність, і середня точність обчислюються для конкретного запиту q. Якщо вибірка велика і розмічена для багатьох запитів, то значення метрик усереднюються по всім запитам.

Другий підхід до вимірювання якості ранжування - це метрика DCG (discounted cumulative gain). Вона використовується в більш складній ситуації, коли оцінки релевантності у можуть бути речовими. Тобто для кожного документа тепер існує градація між релевантністю і не релевантністю. Інші позначення залишаються тими ж, що і для попередньої метрики.

$$\text{DCG@}k(q) = \sum_{i=1}^{k} \frac{2^{y\left(q, d_q^{(i)}\right)} - 1}{\log(i+1)}.$$

Формула для обчислення DCG:

Метрика - це сума дробів. Чим більше релевантним документ, тим більше чисельник в дробу. Знаменник залежить від позиції документа, він штрафує за те, де знаходиться документ. Якщо документ дуже релевантний, але займає низьку позицію, то штраф буде великим, якщо документ релевантний і знаходиться у верхній частині списку, штраф буде маленьким. Таким чином, метрика DCG враховує і релевантність, і позицію документа.

Вона досягає максимуму, якщо все релевантні документи знаходяться в топі списку, причому відсортовані за значенням y.

Дану метрику прийнято нормувати: $$n\mathrm{DCG@}k(q) = \frac{\mathrm{DCG@}k(q)}{\max \mathrm{DCG@}k(q)},$$

де *max DCG@k(q)* - значення DCG при ідеальному ранжируванні.

Після нормування метрика приймає значення від 0 до 1.

2.1.1.2.2. Методи ранжування

Всього виділяють три підходи до вирішення задачі ранжування: pointwise (поточковий), pairwise (попарний), listwise (списковий). Далі будуть наведені по одному методу з кожного підходу, щоб можна було скласти уявлення про їх відмінностях та особливостях.

2.1.1.2.2.1. Поточковий підхід

Найпростіший підхід - це поточковий. У ньому ігнорується той факт, що раціональна цільова змінна задається на парах об'єктів, і оцінка релевантності *a(d, q)* оцінюється безпосередньо для кожного об'єкта. Якщо мова йде про завдання ранжування, то нехай асесор поставив якусь оцінку у кожній парі запит, документ. Ця оцінка і буде пророкувати. При цьому ніяк не враховується, що насправді потрібно передбачити порядок об'єктів, а не оцінки. Цей підхід є простим в тому сенсі, що в ньому використовуються вже відомі методи. Наприклад, можна прогнозувати оцінки з використанням лінійної регресії і квадратичної помилки. Відомо, як вирішувати таке завдання, і таким чином буде отримана релевантність. Далі по виходах моделі можна ранжувати об'єкти.

2.1.1.2.2.2. Попарний підхід

У попарному підході використовуються знання про будову цільової змінної. Модель будується мінімізацією кількості дефектних пар, тобто таких, в яких моделлю був передбачений неправильний порядок:

$$\sum_{x_i<x_j} [a(x_j) - a(x_i) < 0] \to \min.$$

На жаль, цей функціонал дискретний (в нього входять індикатори), тому не вийде мінімізувати безпосередньо його. Однак можна діяти так само, як і з класифікаторами: оцінити функціонал зверху. Можна вважати, що різниця між об'єктами a (xj) -a (xi) - це відступ M, і задати деяку гладку функцію $\sum_{x_i<x_j} L(a(x_j) - a(x_i)) \to \min.$. Якщо використовувати функцію як в логістичній регресії $L(M) = \log(1 + e^{-M}),$ то отриманий метод називається RankNet. Потім можна вирішувати завдання, наприклад, за допомогою стохастичного градієнтного спуску.

2.1.1.2.2.3. Listwise-підхід

У методі RankNet крок стохастичного градієнтного спуску для лінійної моделі виглядає наступним чином: $w := w+\eta\frac{1}{1+\exp(\langle w, x_j - x_i\rangle)}(x_j - x_i).$

Це не дуже складна формула, вона залежить від однієї пари об'єктів. Виникає питання, чи можна модифікувати даний метод (а саме формулу кроку) так, щоб мінімізували НЕ вихідний функціонал, оцінює частку дефектних пар, а DCG.

Відповідь на це питання позитивне. Можна помножити градієнт вихідного функціоналу на те, наскільки зміниться NDCG, якщо замінити місцями x_i і x_j:

$$w := w+\eta\frac{1}{1+\exp(\langle w, x_j - x_i\rangle)}(x_j - x_i) \cdot |\Delta\mathrm{NDCG}_{ij}|(x_j - x_i).$$

Виявляється, що при виконанні градієнтного спуску за допомогою даних кроків оптимізується NDCG.

Це емпіричний факт, і він не доведений. Але на практиці NDCG дійсно покращується при вирішенні завдання даним методом.

Існують і інші підходи до оптимізації NDCG, проте в них робиться спроба роботи з функціоналом, що набагато складніше. Вище описаний найпростіший підхід, він називається LambdaRank.

2.1.1.3. Алгоритми та підходи для роботи з геолокаційними даними

Підставою для дослідження алгоритмів та підходів для роботи з геолокаційними даними є те, що для повноцінної розробки функціоналу з групування подій за ознакою географічної локації потрібно дотримуватися певної точності та швидкості в обробці.

2.1.1.3.1. Geohashing

Широта і довгота кодується в число, яке потім кодується в base-32 [5]. Карта розбивається на матрицю розміру 4x8(див. рис 2.4) кожному осередку присвоюється певний символ (alphanumeric).

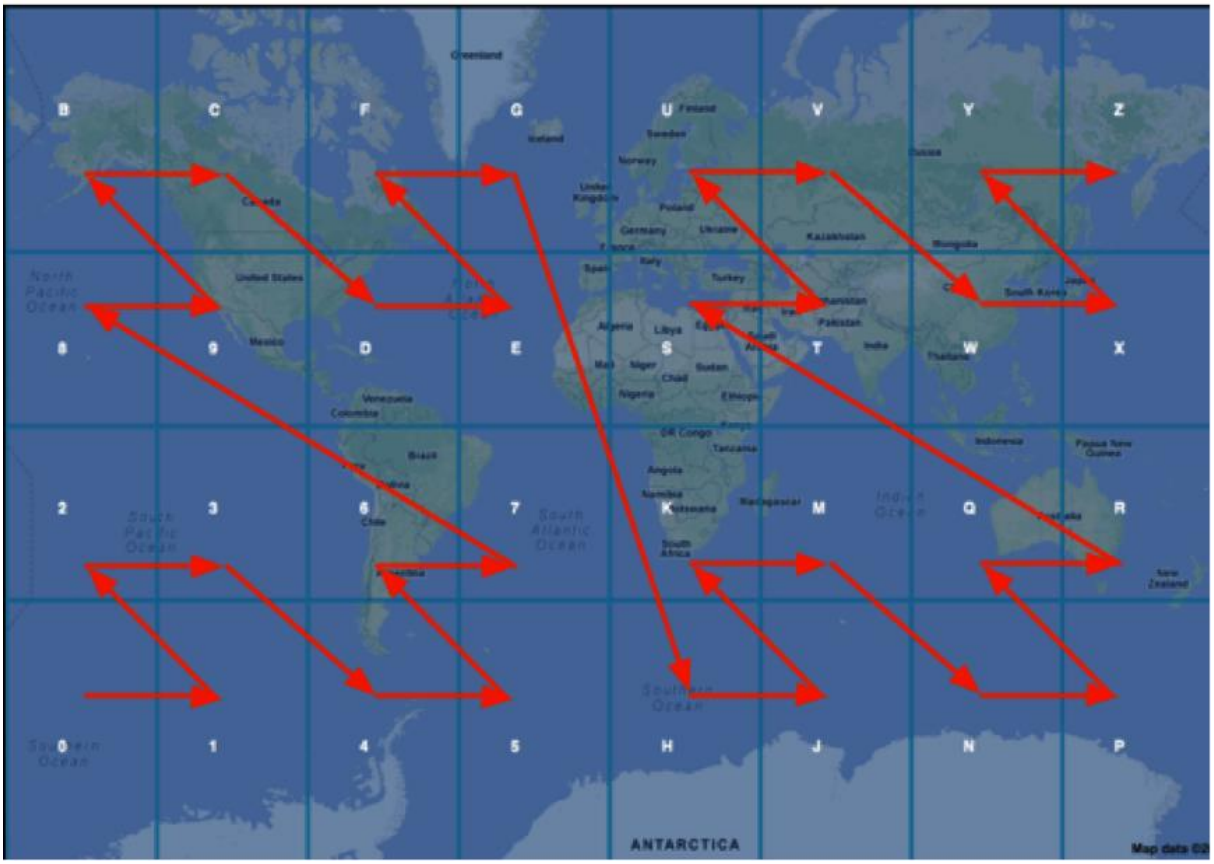


Рис 2.4. Розбиття мапи світу на матрицю.

Щоб підвищити точність, кожна клітинка розбивається на більш дрібні, при цьому до коду додаються символи (якщо бути точним цифри, а після відбувається кодування в base-32).

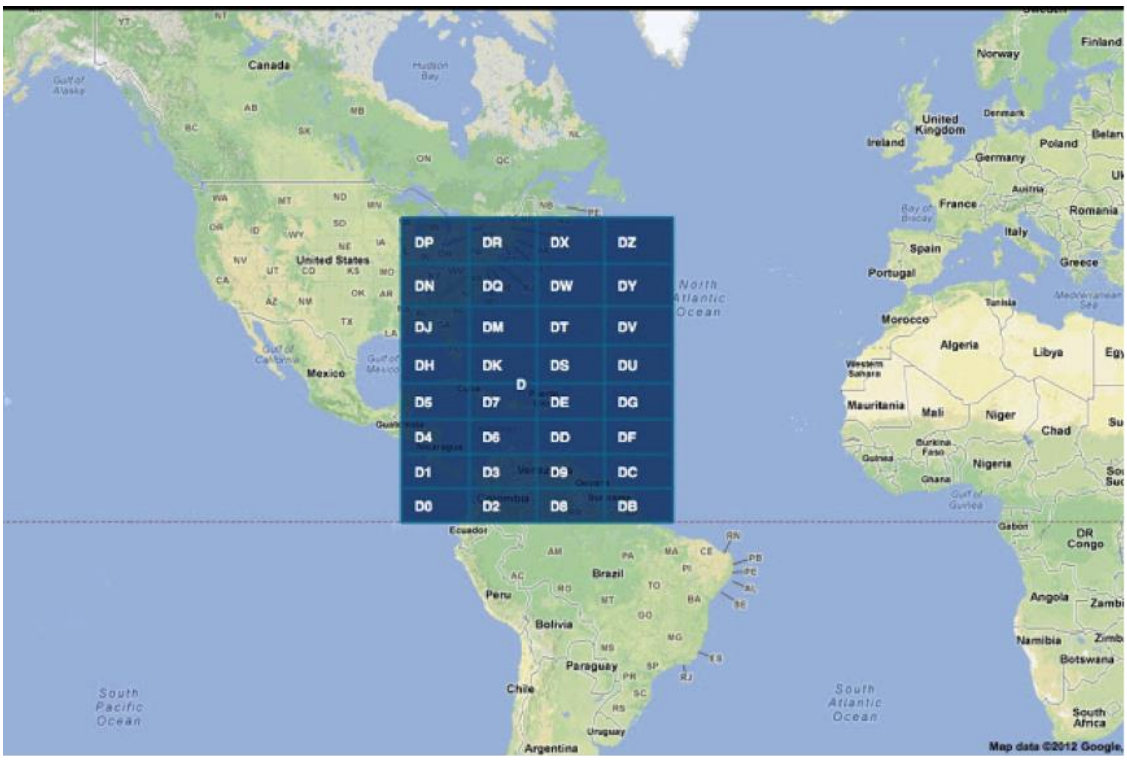


Рис. 2.5. Кожна клітинка матриці розбивається на більш дрібні

Розбиття можна робити до необхідної точності. Такий код унікальний для кожної точки. Його ідея схожа на арифметичне кодування. Даний код звернемо. Приклад: координати 57.64911,10.40744 будуть закодовані в u4pruydqqvj (11 символів). Якщо потрібна менша точність, то і код буде менше. Особливість даного коду в тому, що зазвичай прилеглі точки мають однаковий префікс. І можна порахувавши різницю між гео-хешем визначити близькість двох точок. Але на жаль цей алгоритм не точний, це добре видно з попередніх зображень. Осередки з кодами 7 і 8 знаходяться далі один від одного, ніж осередки 2 і 8. Як приклад приведене зображення (див. рис. 2.6.), де гео-хеш дає невірний результат (geohash delta - різниця між гео хешем без base32)

Рис. 2.6. Гео-хеш дає неправильний результат значення geohashdelta

Якщо точністю в задачі можна знехтувати, то можна створити в таблиці поле geohash, додати по ньому індекс та здійснювати пошук за логарифм log (N).

2.1.1.3.2. Повний перебір

Можна використати для наших цілей процедуру на sql сервері:

create or replace function gc_dist(_lat1 float8, _lon1 float8, _lat2 float8, _lon2 float8) returns float8 as

$$

DECLARE radian CONSTANT float8 := PI()/360;

BEGIN

*return ACOS(SIN($1*radian) * SIN($3*radian) + COS($1*radian) * COS($3*radian) * COS($4*radian-$2*radian)) * 6371;*

END;

$$ LANGUAGE plpgsql;

І використовувати її у запиті :

*SELECT *, gc_dist(54.838971, 83.106560, lat, lng) AS pdist FROM geozones WHERE applicationid = 3890 ORDER BY pdist ASC LIMIT 10;*

Втім, якщо проанлізувати даний запит, то виявиться, що він використовує Seq Scan, що є не найефективнішим варіантом.

Limit (cost=634.72..634.75 rows=10 width=69)

-> Sort (cost=634.72..639.72 rows=2001 width=69)

Sort Key: (gc_dist(54.838971::double precision, 83.10656::double precision, (lat)::double precision, (lng)::double precision))

-> Seq Scan on geozones (cost=0.00..591.48 rows=2001 width=69)

2.1.4.1.3. PostGis

PostGIS - це розширення, яке значно розширює обробку географічних об'єктів в РСУБД PostgreSQL [6].

Для вирішення поставленої задачі буде використана тривимірна система координат SRID 4326 (WGS 84). Дана система координат визначає координати відносно центру мас Землі, похибка становить менше 2 см.

Для початку створимо таблицю з індексом за полем location:

```
CREATE TABLE geozones_test (
uid SERIAL PRIMARY KEY,
lat DOUBLE PRECISION NOT NULL CHECK(lat > -90 and lat <= 90),
lng DOUBLE PRECISION NOT NULL CHECK(lng > -180 and lng <= 180),
location GEOMETRY(POINT, 4326) NOT NULL
);
CREATE INDEX geozones_test_location_idx ON geozones_test USING GIST(location);
```

Після чого для пошуку найближчої геозони можна скористатися наступним запитом :

```
SELECT*,ST_Distance(location::geography, 'SRID=4326;POINT(83.106560 54.838971)'::geography)/1000 as dist_km
FROM geozones_test ORDER BY location <-> 'SRID=4326;POINT(83.106560 54.838971)' limit 10;
```

Тут <-> - distance operator. Ми порахували дистанцію і знайшли найближчі 10 геозон. За логікою данний запит повинен переглянути всі записи в таблиці і порахувати відстань до кожної геозон, що давало би складність O (n).

Втім якщо подивитися explain даного запиту то отримаємо index scan:

Limit (cost=0.00..40.36 rows=10 width=227) (actual time=0.236..0.510 rows=10 loops=1)

-> Index Scan using geozones_test_location_idx on geozones_test (cost=0.00..43460.37 rows=10768 width=227) (actual time=0.235..0.506 rows=10 loops=1)

Order By: (location <-> '0101000020E6100000F4C308E1D1C654406EA5D766636B4B40'::geometry)
Total runtime: 0.579 ms

Причина того, що використовувався index scan у тому, що ми використали четвертий тим індексів які підтримує PostgreSQL - індекси GiST(Generalized Search Trees - узагальнені дерева пошуку) - являє собою якусь інфраструктуру, в якій можуть бути реалізовані багато різних стратегій індексування. GiST-індекси поділяють дані на об'єкти по одну сторону (things to one side), пересічні об'єкти (things which overlap), об'єкти всередині (things which are inside) і можуть бути використані для багатьох типів даних, включаючи дані ГІС.
GiST-індекс реалізований PostGIS підтримує distance operator <-> при пошуку. Також даний індекс може бути складовим. Даний функціонал можна реалізувати і без використання PostGIS, скориставшись індексом btree-gist, але PostGIS надає зручні методи для перекладу широти і довготи в WGS 84.

2.2. Аналіз та порівняння методів розробки баз даних

База даних – це система огранізованого, структурованого зберігання даних, для цифрових систем, як правило на електронних носіях.
У сучасному світі розробки, де більшість додатків оперують інформацією, бази даних є невід’ємною частиною. Для управління базами даних існують СУБД – системи управління базами даних, вони надають необхідний прошарок абстракції за допомогою якого користувач через певний інтерфейс може оперувати базою даних. Також СУБД надають такі розширені можливості як документація змін, відновлення даних після збоїв, резервне копіювання даних.
Бази даних можуть бути поділені за моделями, керуючись якими вони зберігають інформацію:

- Ієрархічна модель предстваляє собою дерево елементів, де у кожного елемента може бути предок і нащадок. Разом з тим у певних елементів можуть бути відсутні або предок або нащадок.

- Мережева модель має схожу структуру на ієрархічну, проте відрязняється тим, що у будь-якого елемента структури може бути більш ніж один предок.
- Реляційна модель передбачає собою структуру у вигляді таблиць. Кожна таблиця має первинний ключ за яким розрізняється унікальність запису в таблиці. Таблиці можуть бути зв’язані між собою так званими зовнішніми ключами.
- Об'єктно-орієнтована модель дозволяє створити структуру бази у вигляді об’єктів.

2.3.1. SQL

SQL (англ. Structured query language - «мова структурованих запитів») - мова програмування, застосовувана для зчитування, додавання, видалення чи модифікаціїї даних, що лежать у реляційній базі даних. Варто відзначити, що певні конструкції у синтаксисі SQL можуть відрізнятися або бути взагалі відсутні у різних реалізаціях СУБД. Втім базові конструкціїї є переважно унифікованими. Більшість СУБД мають підримку таких інструментів як збережені процедури, індексування, вигляди.

Найбільш розповсюдженими представниками є :

- MySQL – відносно проста СУБД, що має найбільшу популярність у середовищі початківців. Відома своїм простим та лаконічним SQL синтаксисом.
- OracleDB – платна реалізація реляційної СУБД від компаніїї Oracle. Має такі потужні інструменти як аналітичні функції та велику кількість оптимізацій.
- PostgreSQL[7] – найбільш популярна реляційна СУБД з відкритим програминим кодом. Розробляється спільнотою програмістів на некомерційній основі. Не дивлячись на свою безкоштовність, має велику кількість оптимізацій та багатий перелік розширень для роботи зі специфічними даними, наприклад геолокацією

- H2 – проста та інтуітивна СУБД, що гарно підходить для локальної розробки через простоту налаштувань та роботу в операційній пам’яті за умовчанням.

2.3.2. NoSQL

NoSQL (від англ. Not only SQL - не тільки SQL) - є не мовою програмування а переліком підходів до зберігання даних у відмінному від SQL форматі. Ця парадигма є конкуруючую для реляційних рішень оскільки пропонує кардинально інші рішення для вирішення проблем, що можуть виникнути при проектуванні та взаємодіїї з базою даних. В той час як SQL-бази даних роблять ставку на структурність, NoSQL фокусуються на атомарності та простоті взаємодії. Осовоположною ідєю є те, що кожен об'єкт при записі до бази даних зберігається у вигляді документу, формат залежить від конкретної реалізації, втім зазвичай це json.

Такий формат взаємодії з базою даних дозволяє абстрагуватися від структури кожного об’єкту оскільки перелік його характеристик може бути будь-яким. Це може бути зручно при розробці системи, що дуже динамічно розвивається і її сутності постійно еволюціонують. При цьому документи можуть посилатися на інші, як це зроблено в реалізації з таблицями у реляційних СУБД, втім такий підхід небажаний, через можливі проблеми з атомарністю та зменшенням швидкості взаємодіїї з СУБД.

Найпопулярнішими реалізаціями NoSQL СУБД є :

- Redis – сховище ключ/значення, яке спрямовано на кешування даних у пам’яті операційної системи. Спрямовано на пришвидшення великомаштабних проектів при умові обробки ними великих обсягів інформації.
- MongoDB – документ-орієнтована база даних, що застосовується у розподілених проектах
- ElasticSearch – пошукова система, що спеціалізується на повнотекстовому пошуку та швидкій індексації великих обсягів тексту.

ВИСНОВКИ ДО РОЗДІЛУ 2

Проаналізувавши відомі технічні та алгоритмічні рішення було прийнято рішення про використання PostGIS технології з деякими покращеннями для видачі результатів за геолокацією. У якості бази даних було обрано реляційну СУБД PostgreSQL. Така комбінація дозволить досягти поставленого завдання з найменшими витратами, при цьому достягнувши результату.

Для ранжування було зроблено висновок про застосування Listwise методу ранжування після досягнення певної кількості користувачів та подій які зберігає та оброблює платформа.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ

3.1. Основні рішення з реалізації

Серверна частина написана с використанням мови програмування Java. Має вигляд RESTful - API веб-сервісу на базі веб-фреймворку Spring 5[8], що приймає виклики з клієнтської частини у форматі json. Всі токени для доступу на зовнішні ресурси передаються як змінні середовища і не знаходяться в коді програми. Веб-сервіс має функціонал аутентифікації/авторизації розроблений на базі Spring Security з інтеграцією Google Oauth2[9]. Метод розпізнавання цифрового відбитку користувача - Bearer токен.
Для доступу до бази даних використовується ORM Hibernate. Hibernate надає прошарок абстракції який дозволяє в простих операціях абстрагуватися від нативних SQL запитів та працювати з об'єктами. У разі потреби данна реалізація JPA[10] дозволяє виконувати складні нативні SQL запити.
База даних має два інстанси :

- для локальної розробки використовується PostgreSQL Docker-контейнер[11], який запускається к командного рядка, стабільна версія бази даних на публічному середовищі - це інстанс
- PostgreSQL, що запускається за допомогою heroku-плагіна під час кожного розгортання сервісу на публічному середовищі.

Для зберігання зображень виконана інтеграція зі стороннім сервісом Amazon S3 (Simple Cloud Storage Service)[12] - сервіс, що пропонується веб-службами Amazon, яка забезпечує зберігання об'єктів через інтерфейс веб-сервісу. Amazon S3 використовує ту саму масштабовану інфраструктуру зберігання, яку Amazon.com використовує для запуску глобальної мережі електронної комерції. Для запуску веб-сервісу на публічному середовищі обрано Heroku[13] - хмарна платформа як сервіс, що підтримує кілька мов програмування. Одна з перших хмарних платформ, Heroku розробляється з

червня 2007 року, коли вона підтримувала лише мову програмування Ruby, але тепер підтримує Java, Node.js, Scala, Clojure, Python, PHP та Go.

Технології для розробки були використані такі :

- Серверна частина: Java 11, Spring Boot 2.4, Spring MVC, Spring Security, Tomcat 8, Docker, PostgreSQL, JPA/ Hibernate, Google Oauth, Heroku, Amazon AWS S3, Lombok, PostGIS.
- Середовище розробки: Intellij Idea (студентська версія).

Весь хід розробки було задукоментовано завдяки системі версіонування коду Git. Платформа версіонування коду - GitHub. Життєвий цикл розробки побудований на основі git flow(див рис. 3.1.). Після злиття коду у гілку master виконувалося завантаження програми на робоче оточення.

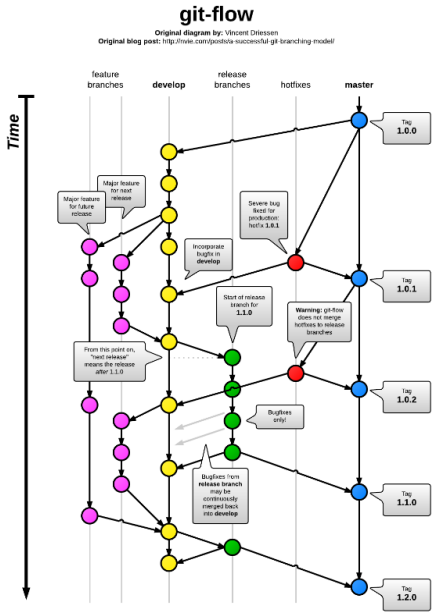


Рис 3.1. Приклад схеми для злиття коду у віддалений репозиторій

3.2. Архітектура системи

Архітектура платформи(див. рис. 3.2.) являє собою монолітний веб-сервіс, що комунікує з базою даних, реалізує бізнес-логіку платформи та виконує інтеграції з зовнішніми системами.

Наведена нижче схема ілюструє високорівневу архітектуру сервісу. Вона складається з таких компонентів:

- Клієнтські додатки – додатки, що комунікують з сервісом та надають користувачам доступ до функціоналу. Підтримуються додатки для будь-якої існуючої операційної системи.
- Веб-сервіс – серверний додаток, що реалізує логіку системи.
- База даних – сховище для даних, якими оперує система. Веб-сервіс має змогу для зміни провайдера СУБД, втім основним обрано реляційну СУБД PostgreSQL.
- Хмарне сховище – середовище в рамках якого розгортається сервер системи та сервер бази даних.
- Зовнішня система зберігання мультимедіа – зовнішній сервіс для зберігання зображень, відео. У якості реалізації було обрано Amazon S3 Storage.

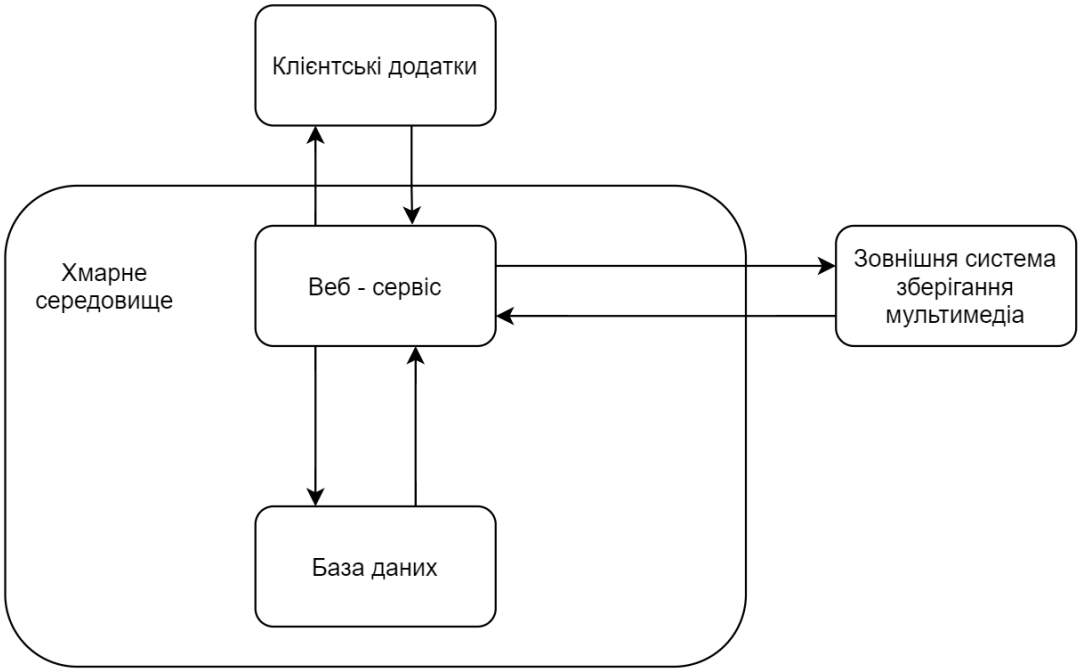


Рис 3.2. Високорівнева архітектура системи

3.3. Розробка алгоритмів та баз даних

У системі використовується PostGIS технології з деякими покращеннями для видачі результатів за геолокацією.

Для зберігання даних було обрано реляційну СУБД PostgreSQL з наступних причин :

1. Відкритий код
2. Безкоштовне використання
3. Потужність вбудованих алгоритмів

Пошук за геолокацією побудований на алгоритмі PostGIS. Це розширення, яке значно розширює обробку географічних об'єктів в СУБД PostgreSQL.

Для вирішення нашої задачі налаштовано тривимірну систему координат SRID 4326 (WGS 84). Дана система координат визначає координати відносно центру мас Землі, похибка становить менше 2 см.

3.3.1. Проектування схеми бази даних

Проектування схеми бази даних відбувалося відштовшуючись від задач які ставляться перед продуктом і моделі даних.

Зі схеми можна виділити основні таблиці:

- Events_table
- Images_table
- Users_table
- Users_table_subscribers
- Users_table_subscriptions
- Refresh_token

3.4. Розробка системи

Повна екосистема системи складається з двох взаємодіючих частин, що спілкуються між собою за http протоколом:

1. Серверний додаток, що відображає бізнес-логіку системи та відповідає за зв’язок системи з базою даних.

2. Клієнтський додаток, що дає користувачу доступ до системи та виступає інтерфейсом для зв’язку з серверною частиною.

Архітектура клієнтської частини Model View Controller :

1. Реалізація користувальницького інтерфейсу в коді.
2. Використання layout anchors.
3. Компоненти (кожен з яких виконує одну конкретну функцію)
4. Поділ призначеного для користувача інтерфейсу і логіки.

3.4.1. Модель даних

Модель програми - це дані якими вона оперує, що організовані у певні сутності. У нашому випадку кожною сутністю є Java - об'єкт[14]. Основні сутності:

- User. Відображає користувача, який пройшов реєстрацію. Може бути адміністратором або звичайного користувача.
- Event. Об’єкт що відображає подію. Має в собі необхідну інформацію що повинна містити подія.
- Image. Відображає зображення. Зв’язана з подією та користувачем.
- RefreshToken. Містить в собі поле refresh_token, дає можливість зробити алгоритм логіну та реєстрації зручним та безпечним.

3.4.2. Модель вхідних даних

Вхідні дані передаються у форматі json, потім десеріалізуються за допомогою бібліотеки jackson. Приклад вхідних даних для реєстрації:

```
{
"email":"andrey.belyi1332@gmail.com",
"firstName":"Славик1123",
"secondName":"Винницкий112",
"login":"slavick23211212",
"password":"slavick2321",
"confirmPassword":"slavick2321",
```

"image": "https://s3.eu-west-3.amazonaws.com/events--core-images/1569067282021-flyby_1920.jpg"
}

3.4.3.Модель вихідних даних

Вихідні дані серіалізуються у формат json, за допомогою бібліотеки jackson. Приклад вихідних даних для додавання події користувачем:

{
"id": 36,
"title": "Выставка автомобилей",
"description": "Приглашаю всех на выставку автомобилей в городе-львве",
"imagesLinks": [
"https://s3.eu-west-3.amazonaws.com/events--core-images/1589732687551-unnamed.jpg"
]
},
"creator": {
"id": 1,
"firstName": "Славик",
"secondName": "Винницкий",
"subscribers": 1,
"subscriptions": 1,
"subscribed": true
}
}

3.4.4. Функціонал користувача

Функіонал користувача – це набір можливостей які підтримує система в роботі з частиною моделі данних, а саме User (далі користувач). Вона є основоположною у системі оскільки на ній базується весь додаток. Наприклад для того, щоб реалізувати авторизацію та аутентифікацію потрібно оперувати якоюсь моделлю. Користувач і є цією моделюю у даному випадку. Користувачі можуть бути двох виглядів: звичайний користувач та адміністратор.

3.4.4.1. Логін

Логін доступний за адресою */auth/login*. Обновлення токену доступу знаходиться за адресою */auth/token*. Користувач логіниться в додатку, передаючи логін / пароль і fingerprint пристрою. Сервер перевіряє достовірність логіну й пароля і у разі успіху створює і записує сесію в базу даних:

```
{
userId: uuid,
refreshToken: uuid,
expiresIn: int,
fingerprint: string,
}
```

Відправляє клієнту два токена access і refresh token uuid (взятий з вище створеної сесії) :

```
{
accessToken : eyJhbGciOiJIUzI1NiIs ...,
refreshToken : 9f34dd3a-ff8d-43aa-b286-9f22555319f6
}
```

Клієнт зберігає токени (access в пам'яті програми, refresh персистентно), використовуючи access token для подальшої авторизації запитів.

Варто зауважити що процес додавання сесії в таблицю повинен мати свої заходи безпеки тому при додаванні перевіряється скільки сесій всього є у користувача і якщо їх занадто багато або юзер підключається одночасно з декількох підмереж, варто вжити заходів. Імплементуючи дану перевірку я перевіряю тільки, щоб користувач мав максимум до 5 одночасних сесій максимум, і на 6'ій видаляю всі інші сесії крім поточної (6'ї).

Таким чином якщо користувач залогінився на п'яти пристроях, рефреш токени будуть постійно оновлюватися і всі щасливі. Але якщо з аккаунтом користувача почнуть виробляти підозрілі дії (спробують ввійти з більш ніж 5'ті пристроїв) система скине всі сесії (рефреш токени) крім останньої.

Перед кожним запитом клієнт попередньо перевіряє час життя access token (та беремо expires_in прямо з JWT в клієнтському додатку) і якщо воно минув використовує refresh token щоб оновити обива токена і продовжує використовувати новий access token. Для більшої впевненості можемо оновлювати токени на кілька секунд раніше.

Fingerprint використовується як інструмент відстеження пристрою незалежно від бажання користувача бути ідентифікованим. Це хеш згенерований на базі деяких унікальних параметрів / компонентів клієнтського пристрою. Перевага fingerprint в тому, що він ніде персистентно не зберігається і генерується тільки в момент логіну і рефрешу.

Для використання можливості аутентифікації на більш ніж одному девайсі необхідно зберігати всі рефреш токени по кожному користувачеві. Збереження цього списку в PostgreSQL таблиці[15]. У процесі кожного логіна створюється запис з IP / Fingerprint та іншої мета інформацією тобто сесія.

1. Клієнт перевіряє перед запитом не минуло чи час життя access token
2. Якщо минув клієнт відправляє на *auth/refresh-token*:

   {
   refreshToken: uuid,
   fingerprint: string
   }

3. Сервер отримує запис сесії по UUID рефреш токена
4. Зберігає поточну сесію в змінну і видаляє її з таблиці
5. Перевіряє поточну сесію:
   a. Чи не минув чи час життя
   b. На відповідність старого fingerprint отриманого з поточної сесії з новим отриманим з тіла запиту
6. У разі негативного результату повертає помилку INVALID_SESSION
7. У разі успіху створює нову сесію і записує її в БД
8. Створює новий accessToken
9. Відправляє клієнту { accessToken, refreshToken }

3.4.4.2. Реєстрація

Реєстрація у платформі доступна за адресою */auth/register*. Точка реєстрації є відкритою для анонімних користувачів, тобто тих, що ще не пройшли процедуру логіну або реєстрації.

При реєстрації клієнт збирає данні у користувача у вигляді форми, передає їх у об’єкті json :

```
{
email:andrey.belyi2@gmail.com,
firstName:Славик,
secondName:Винницкий,
login: andrey.belyi2,
password:andrey.belyi2,
confirmPassword:andrey.belyi2,
image: https://s3.amazonaws.com/events-images/1569067282021.jpg
}
```

Після чого сервер валідує унікальні поля, такі я login, password, confirmPassword, email та у разі успіху ствроює запис користувача та сесії.

3.4.4.3 Пошук користувача

Пошук за користувачами доступний за адресою */users/find.* Для створення соціального оточення кожного користувача основним є функціонал пошуку інших користувачів та переглядати їх профілі. Для цього за маршрутом /api/users/find реалізовано публічний метод, що сприймає опціональні параметри login, first name, last name, email за якими здійснюється пошук користувача в базі даних.

Також цей метод підтримує пагінацію - спосіб розбити пошукові запити на частини для полегшення навантаження на систему.

Підтримку пагінації та сортування надають такі параметри:

- limit
- page
- sort

Наприклад, якщо викликати метод з параметрами limit=12, page=0, sort=login,desc, то у результаті отримаємо перші 12 записів відсортовані по полю login у зворотньому порядку.

3.4.4.4. Особистий кабінет

Особистий кабінет доступний за адресою /*users/me* для перегляду своєї сторінки і */users/{id}* для будь-якого іншого користувача.

ВИСНОВКИ ДО РОЗДІЛУ 3

Розроблена система має перелік необхідного функціоналу для залучення аудиторії в додаток. Реалізована підтримка для будь-якого типу клієнтських додатків. Користувачу для можливості користатися платформою потрібно всього лише завантажити клієнтський додаток на свій пристрій та війти в систему. Створена система полегшує процес пошуку заходів, що відбуваються неподалік або ж підбирати захід для відвідування за інтересами. Розроблена система авторизації та аутентифікації надає базові можливості для визначення користувача, його привілегій та пристрою. В майбутньому можна буде розширити систему додаванням нових ролей користувач та адміністратор. Під час розробки системи, котра хоч і реаліує поставлену перед собою задачу, було відкрито багато місць які потребують поліпшення та оптимізації. До процесу пошуку подій можна додати кешування, що візуально пришвидшить роботу системи у рази. Також варто задуматися над міграцією від монолітного веб-сервісу до розподіленого з окремими мікросервісами кожен з яких відповідав би за свою частину доменної моделі. Це дозволило б запускати різні екземпляри одного й того ж сервісу, що істотно збільшило б пропускну спроможність платформи.

РОЗДІЛ 4
ПЕРЕВІРКА РОБОТИ СИСТЕМИ

4.1. Опис основних розроблених кінцевих точок системи

Список розроблених кінцевих точок є досить обширним, тому буде доцільно розглянути основні з них :

1. Логін. Дозволяє отримати токен з яким користувач зможе користуватися функціоналом додатку. Токен видається лише у разі, якщо користувач уже зареєстрований у системі.
2. Реєстрація. Користувач вводить свою персональну інформацію таку як ім’я, прізвище, логін, пароль, електронну пошу додаючи свої дані в систему. Після цього він зможе пройти процедуру логіна і користуватися платформою. Уся чутлива інформація, наприклад паролі, хешується і не зберігається у прямому вигляді у базі даних.
3. Отримання подій. Кінцева точка, що зазвичай використовуються як найбільш вживана. Надає можливість клієнту отримати всі події, або певну задану кількість, відсортувати їх. За умовчуванням шукає події, що ще не почалися.
4. Додавання події. Точка яка є основним джерелом наповнення системи контентом.
5. Підписка і відписка на користувача. Соціальна функція, що дозволяє слідківати за певними користувачами, що створюють заплюючі на вашу думку події.
6. Відвідування події. Відвідування події дає змогу організатору мати змогу оцінити яка кількість людей прийде на захід, який він створив.
7. Пошук події. Якщо вас цікавить якась конкретна подія, або конкретного автора ви можете створити пошуковий запит, який буде оброблено і у результаті якого ви отримаєте список заходів з характеристиками які вас цікавлять.

4.2. Заміри часу на відповідь основних ендпоінтів

Для перевірки роботи системи було розгорнуто тестове середовище з фіксованою кількістю даних і виконано 100 послідовних запитів на кожну із кінцевих точок (*див. табл. 4.1*.).

Табл. 4.1

№ з/п	Назва кінцевої точки	Адреса	Середній час відповіді
1	*Логін*	*/auth/register*	*190 мс*
2	*Реєстрація*	*/auth/login*	*163 мс*
3	*Обновлення токену*	*/auth/token*	*54 мс*
4	*Пошук користувача*	*/api/users/find?q=Slavick*	*80 мс*
5	*Отримання підписок користувача*	*/api/users/1/subscriptions*	*95 мс*
6	*Отримання підписників користувача*	*/api/users/1/subscribers*	*120 мс*
7	*Підписка/відписка на користувача*	*/api/subscribe/1*	*113 мс*
8	*Особистий кабінет користувача*	*/api/users/1*	*43 мс*
9	*Свій особистий кабінет*	*/api/users/me*	*21 мс*
10	*Отримання списку користувачів*	*/api/users/*	*195 мс*
11	*Отриманнання функцій*	*/api/features*	*135 мс*
12	*Отримання подій*	*/api/events*	*220 мс*
13	*Відображення подій на карті*	*/api/maps/events*	*243 мс*
14	*Додавання події*	*/api/events/add*	*215 мс*
15	*Отримання подіїї*	*/api/events*	*160 мс*
16	*Пошук події*	*/api/events/find?*	*240 мс*
17	*Отримання відвідувачів події*	*/api/events/1/visitors*	*219 мс*
18	*Обновлення події*	*/api/events/update*	*235 мс*
19	*Зацікавитися у події*	*/api/events/like/1*	*176 мс*
20	*Відвідати подію*	*/api/events/visit/1*	*156 мс*
21	*Видалити подію*	*/api/events/delete/1*	*200 мс*
22	*Отримання зображення*	*/api/getFile*	*34 мс*
23	*Додавання зображення*	*/api/uploadFile*	*125 мс*
24	*Видалення зображення*	*/api/deleteFile*	*174 мс*

4.3. Порівняння системи з аналогами

Протягом виконання бакалаврської дипломної роботи було переглянуто велику кількість програм та веб-додатків, що мають схожий функціонал тому доцільно провести порівняння між існуючими аналогами та системою, що була реалізована в результаті виконання роботи. Загалом, ті додатки, що фокусуються на елементі соціалізації, наприклад Tinder, NextDoor мало уваги приділяють заходам та подіям, які можуть цікавити людей. З іншої сторони ті

додатки, що роблять акцент на заходах мають слабкий функціонал соціалізаціїї. Майже в жодному додатку немає можливості переглянути на мапі усі уснуючі події, або для початку ті, що знаходяться поруч. Багато з додатків мають за собою мету не спростити і урізноманітнити життя користувача, а монетизувати додаток за рахунок данних користувачів.

ВИСНОВКИ ДО РОЗДІЛУ 4

В останньому розділі було розглянуто та описано розроблений функціонал системи з пошуку подій. Також були заміряні певні метрики, наприклад середній час відповіді кожного з кінцевих точок. Аналізуючи час, який потрібний системі для повернення результатів з кінцевих точок, можна сказати, що швидкодія є прийнятно. Втім для того, щоб система могла оброблювати великі масиви і даних та обслуговувати паралельно великий масштаб користувачів, деякі кінцеві точки можна покращити з точки зоу продуктивності. Почати варто з тих які мають середній час відповіді більше ні 200 мілісекунд. Першими кроками для пришвидшення може бути аналіз запитів, що генеруються об'єктно реляційним фреймворком для доступу до бази даних. Після цього можна проаналізувати кількість взаємодій які мають між собою різні частини доменної моделі. Наступними кроками може бути підключення та налаштування кешування другого рівня яке дасть змогу в деяких випадках звертатися до бази даних набагато рідше. Також систему було порівняно з існуючими аналогами, в результаті чого дана система мала переваги над деякими своїми конкурентами.

ВИСНОВКИ

У даній роботі була розглянута проблема створення додатку який би допомагав людям з процесом пошуом заходів, що відбуваються неподалік або ж підбирати захід для відвідування за інтересами.

За результатами роботи було розлоблено платформу, яка дозволяє публікувати свої події, відстежувати кількість людей, що зацікавилися у ній, знаходити заходи, що будуть цікаві кожному.

Платформа, що була розроблена, має кілька переваг над конкурентними сервісами:

- безкоштовність – усі функції платформи є повністю і назавжди безоплатними у використанні, що робить її доступною для будь-кого.
- доступність - доступ до додатку можливий з будь-якої платформи на яку написаний клієнтський додаток.
- безпечність - додаток не збирає і не оброблює персональних даних, не монетизується за рахунок користувача.

У роботі було проведено дослідження ринку з існуючих рішень, проаналізовані алгоритми, що стали б у пригоді при розширенні сервісу. Система була інтегрована з мобільним додатком на операційній системі IOS.

ПЕРЕЛІК ПОСИЛАНЬ

1. Кластерный анализ – Режим доступу: https://en.wikipedia.org/wiki/Cluster_analysis
2. K-Means Clustering – Режим доступу: https://en.wikipedia.org/wiki/K-means_clustering
3. Алгоритми пошукової системи «Яндекс» – Режим доступу: https://pixelplus.ru/samostoyatelno/stati/prodvizhenie-saytov/algoritmy-ranzhirovaniya-yandex.html
4. Berners-Lee Tim HyperText Transfer Protocol // World Wide Web Consortium, 2010.
5. Работа с геолокациями в режиме highload – Режим доступу: https://habr.com/ru/post/228023/
6. Геолокаційні запити в PostgreSQL без важкої артилерії – Режим доступу: https://dou.ua/lenta/articles/geolocation-requests-postgresql/
7. PostgerSQL About – Режим доступу: https://www.postgresql.org/about/
8. Spring Boot Reference Documentation – Режим доступу: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
9. Google Oauth – Режим доступу: https://www.oauth.com/oauth2-servers/openid-connect/
10. Pro JPA 2 in Java EE 8 An In-Depth Guide // Keith, Mike, Schincariol
11. About Docker – Режим доступу: https://www.docker.com/why-docker
12. Amazon AWS S3 Documentation – Режим доступу: https://aws.amazon.com/s3/storage-classes/
13. What is Heroku – Режим доступу : https://devcenter.heroku.com/articles/getting-started-with-java
14. Pro Spring 5: An In-Depth Guide 5th Edition // Iuliana Cosmina
15. Spring Data JPA - Reference Documentation [Електронний ресурс]. – Spring Framework Reference Documentation, 2020. – Режим доступу: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference.

ДОДАТОК А. Код програми.

1. Екземпляр створеного контролеру що відповідає за ендпоінт події:

```
@RestController
@RequestMapping("/api")
@ConditionalOnProperty(name = "features.events.common")
public class EventController {

    @Autowired
    private EventService service;

    @Autowired
    private UserService userService;

    @GetMapping("/events")
    public List<EventDto> getEvents(Pageable pageable) {
        return service.findAllFutureEvents(pageable);
    }

    @GetMapping(path = "/events/{id}")
    public EventDto getEvent(@PathVariable("id") long eventId) {
        return service.find(eventId);
    }

    @GetMapping(path = "/events/find")
    public List<EventDto> searchEvents(@RequestParam("q") String search, Pageable pageable)
{
        return service.searchEventLIKEGOOGLE(search, pageable);
    }

    @GetMapping(path = "/events/{id}/visitors")
    public List<SmallUserDto> getEventVisitors(@PathVariable("id") long eventId) {
        return service.getEventVisitors(eventId);
    }

    @PutMapping("/events/like/{id}")
    public EventDto likeEvent(@PathVariable(name = "id") long eventId) {
        User user = userService.getRequester();
        return service.likeEvent(user, eventId);
    }

    @PutMapping("/events/visit/{id}")
    public EventDto visitEvent(@PathVariable(name = "id") long eventId) {
        User user = userService.getRequester();
        return service.visitEvent(user, eventId);
```

```
    }

    @PostMapping("/events/add")
    public EventDto createNewEvent(@RequestBody Event event) {
        return service.save(event);
    }

    @PutMapping(path = "/events/update/{id}")
    public EventDto updateEvent(@PathVariable("id") long eventId, @RequestBody Event
newEvent) {
        return service.update(eventId, newEvent);
    }

    @DeleteMapping(path = "/events/delete/{id}")
    public void deleteEvent(@PathVariable("id") long id) {
        service.delete(id);
    }

}
```

2. Екземпляр классу рівня сервіс для обробки бізнес-логіки події.

```
@Service
public class EventService {

    @Autowired
    private EventMapper eventMapper;

    @Autowired
    private EventRepository eventRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private ImageService imageService;

    @Autowired
    private UserMapper userMapper;

    @Autowired
    private UserService userService;


    @Transactional
    public List<EventDto> findAllFutureEvents(Pageable pageable) {
```

```
        User userRequester = userService.getRequester();
        User persistedUser = userRepository.findById(userRequester.getId())
                .orElseThrow(UserNotFoundException::new);

        return eventRepository.findAllByStartDateGreaterThan(LocalDateTime.now(), pageable)
                .stream()
                .map(event -> eventMapper.convertToEventDto(event, persistedUser))
                .collect(Collectors.toList());
    }

    @Transactional
    public EventDto save(Event entity) {
        List<Image> eventImages = entity.getImages().stream()
                .map(image -> imageService.find(image.getId()))
                .collect(Collectors.toList());
        entity.setImages(new HashSet<>(eventImages));
        entity.setCreationDate(LocalDateTime.now());

        User creator = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

        User persistedUser = userRepository.findByLogin(creator.getLogin()).orElseThrow(UserNotFoundException::new);
        entity.setCreator(persistedUser);
        return eventMapper.convertToEventDto(eventRepository.save(entity), persistedUser);

    }

    @Transactional
    public EventDto find(long id) {
        return eventMapper.convertToEventDto(
                eventRepository.findById(id)
                        .orElseThrow(() -> new IllegalArgumentException(String.format(WRONG_INDEX, id))),
                userService.getRequester());
    }

    public EventDto update(long id, Event entity) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        session.update(entity);
        session.getTransaction().commit();
        session.close();
        entity.setId(id);
        return eventMapper.convertToEventDto(entity, userService.getRequester());
```

```
    }

    @Transactional
    public void delete(long id) {
        eventRepository.deleteById(id);
    }

    @Transactional
    public List<EventDto> searchEventLIKEGOOGLE(String searchWord, Pageable pageable) {
        User userRequester = userService.getRequester();
        User persistedUser = userRepository
                .findByLogin(userRequester.getLogin())
                .orElseThrow(UserNotFoundException::new);
        return eventRepository.searchEvents(searchWord.toUpperCase(), pageable)
                .stream()
                .map(event -> eventMapper.convertToEventDto(event, persistedUser))
                .collect(Collectors.toList());
    }

    @Transactional
    public List<EventDto> findEventsByLocation(LocationDto leftBotPoint, LocationDto
rightTopPoint) {
        User userRequester = userService.getRequester();
        User                                persistedUser                                =
userRepository.findByLogin(userRequester.getLogin()).orElseThrow(UserNotFoundException
::new);
        return eventRepository
                .findAllByLocation(
                        leftBotPoint.getLatitude(), rightTopPoint.getLatitude(),
                        leftBotPoint.getLongitude(), rightTopPoint.getLongitude())
                .stream().map(event                ->                eventMapper.convertToEventDto(event,
persistedUser)).collect(Collectors.toList());
    }

    @Transactional
    public EventDto likeEvent(User user, long eventId) {
        Event                               eventToLike                                =
eventRepository.findById(eventId).orElseThrow(RuntimeException::new);
        User                                persistedUser                                =
userRepository.findByLogin(user.getLogin()).orElseThrow(UserNotFoundException::new);
        setOrDeleteLike(eventToLike, persistedUser);
        return eventMapper.convertToEventDto(eventToLike, persistedUser);
    }

    @Transactional
```

```
    public EventDto visitEvent(User user, long eventId) {
        Event eventToVisit = eventRepository.findById(eventId)
                .orElseThrow(RuntimeException::new);
        User persistedUser =
userRepository.findByLogin(user.getLogin()).orElseThrow(UserNotFoundException::new);
        setOrDeleteVisit(eventToVisit, persistedUser);
        return eventMapper.convertToEventDto(eventToVisit, persistedUser);
    }

    @Transactional
    public List<SmallUserDto> getEventVisitors(long eventId) {
        return eventRepository
                .findById(eventId)
                .orElseThrow(RuntimeException::new).getVisitors()
                .stream()
                .map(user -> userMapper.convertToSmallUserDto(user))
                .collect(Collectors.toList());
    }

    private void setOrDeleteLike(Event eventToLike, User persistedUser) {
        Set<User> eventLikes = eventToLike.getLikes();
        if (eventLikes.contains(persistedUser)) {
            eventLikes.remove(persistedUser);
        } else {
            eventLikes.add(persistedUser);
        }
    }

    private void setOrDeleteVisit(Event eventToVisit, User persistedUser) {
        Set<User> visitors = eventToVisit.getVisitors();
        if (visitors.contains(persistedUser)) {
            visitors.remove(persistedUser);
        } else {
            visitors.add(persistedUser);
        }
    }
}
```

3. Екземпляр классу-сутності, що відображає частину моделі події :

```
@Data
@Entity
@Table(name = "events_table")
@NoArgsConstructor
@AllArgsConstructor
```

```
@Builder
public class Event {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String title;

    @ManyToOne
    private User creator;

    private LocalDateTime creationDate;

    private LocalDateTime startDate;

    private LocalDateTime endDate;

    @ManyToMany
    @Builder.Default
    private Set<User> likes = new HashSet<>();

    @ManyToMany
    @Builder.Default
    private Set<User> visitors = new HashSet<>();;

    private String description;

    @Embedded
    @AttributeOverrides(value = {
        @AttributeOverride(name = "longitude", column = @Column(scale = 14, precision =
18)),
        @AttributeOverride(name = "latitude", column = @Column(scale = 14, precision = 18))
    })
    private LocationDto location;

    @OneToMany(cascade = {ALL}, fetch = FetchType.EAGER)
    @JoinColumn(name = "image_id")
    private Set<Image> images;

}
```

4. Екземпляр сервісу для обробки успішної аутентифікації:

```
@Component
```

```
public class CustomAuthenticationSuccessHandler extends SimpleUrlAuthenticationSuccessHandler {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication) throws IOException, ServletException {
        System.out.println("token");
            if (response.isCommitted()) {
            return;
        }
            DefaultOAuth2User oidcUser = (DefaultOAuth2User) authentication.getPrincipal();
        Map<String, Object> attributes = oidcUser.getAttributes();
        System.out.println(attributes);
        String email = (String) attributes.get("email");
        User user = userRepository.findByLogin(email).orElseThrow(UserNotFoundException::new);
        if (user == null) {
            user = new User();
            user.setLogin(email);
            user.setPassword("def");
            userRepository.save(user);
        }
        String token = jwtTokenUtil.generateToken(user);
        String redirectionUrl = UriComponentsBuilder.fromUriString("http://localhost:8080/home")
                .queryParam("auth_token", token)
                .build().toUriString();
        getRedirectStrategy().sendRedirect(request, response, redirectionUrl);
    }

}
```

5.Скрипт бази даних:

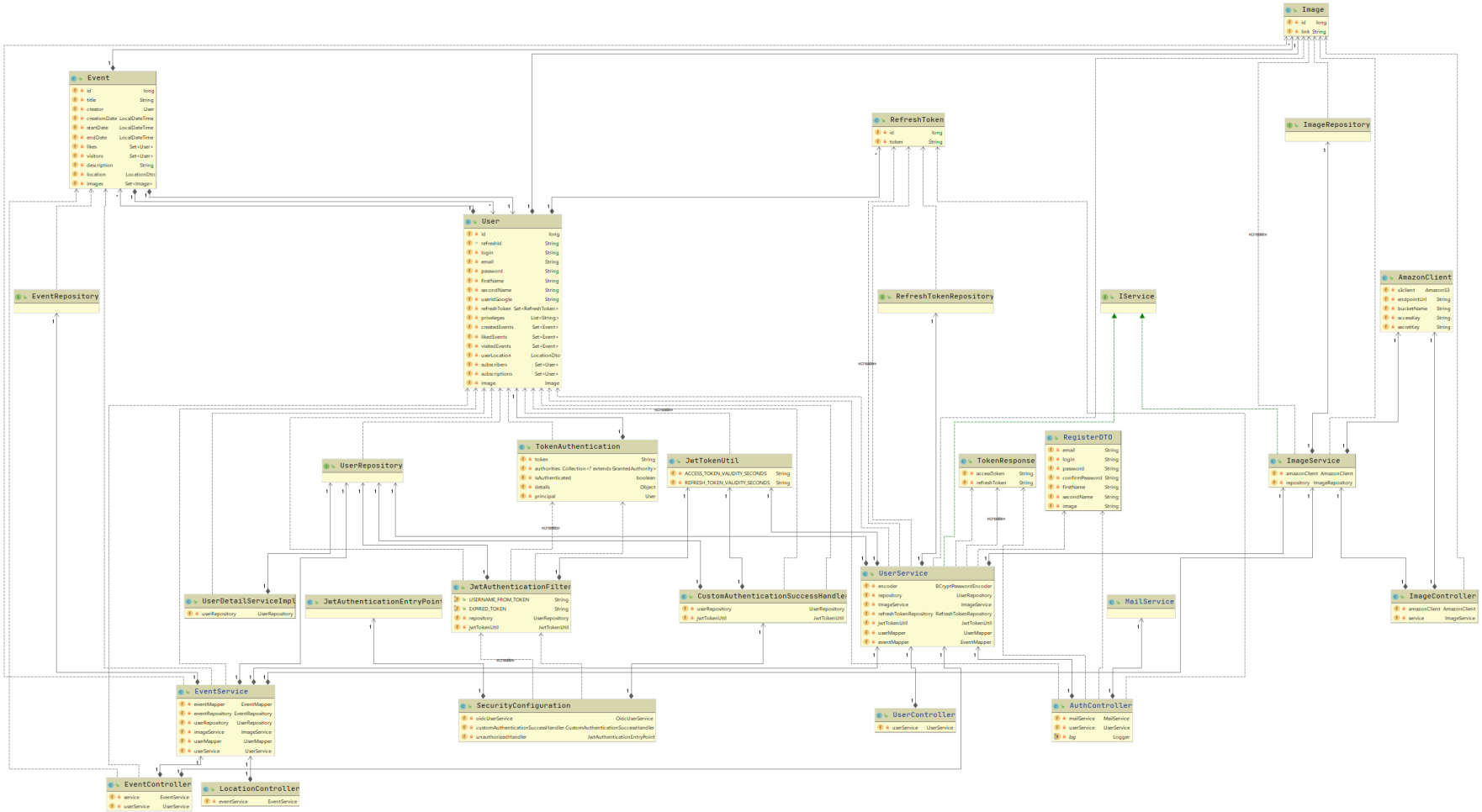
```
 create table users_table_created_events
 (
        user_id bigint not null,
        created_events_id bigint not null
 constraint uk_tdny8rk5t282n2xg87r7kq4lq
 unique,
 constraint users_table_created_events_pkey
```

```
primary key (user_id, created_events_id)
);

alter table users_table_created_events owner to postgres;

create table events_table
(
    id bigserial not null
constraint events_table_pkey
primary key,
    creation_date timestamp,
description varchar(255),
    end_date timestamp,
destination varchar(255),
    latitude numeric(18,14),
longitude numeric(18,14),
    start_date timestamp,
title varchar(255),
    creator_id bigint
);

alter table events_table owner to postgres;

create table images_table
(
    id bigserial not null
constraint images_table_pkey
primary key,
    link varchar(255)
constraint uk_nx53xfg005fkytg0tldktlchk
unique,
image_id bigint
constraint fko4h2ny7q05s11asrbvf44ltb6
references events_table
);

alter table images_table owner to postgres;

create table refresh_token
(
    id bigserial not null
constraint refresh_token_pkey
primary key,
    token varchar(255)
);
```

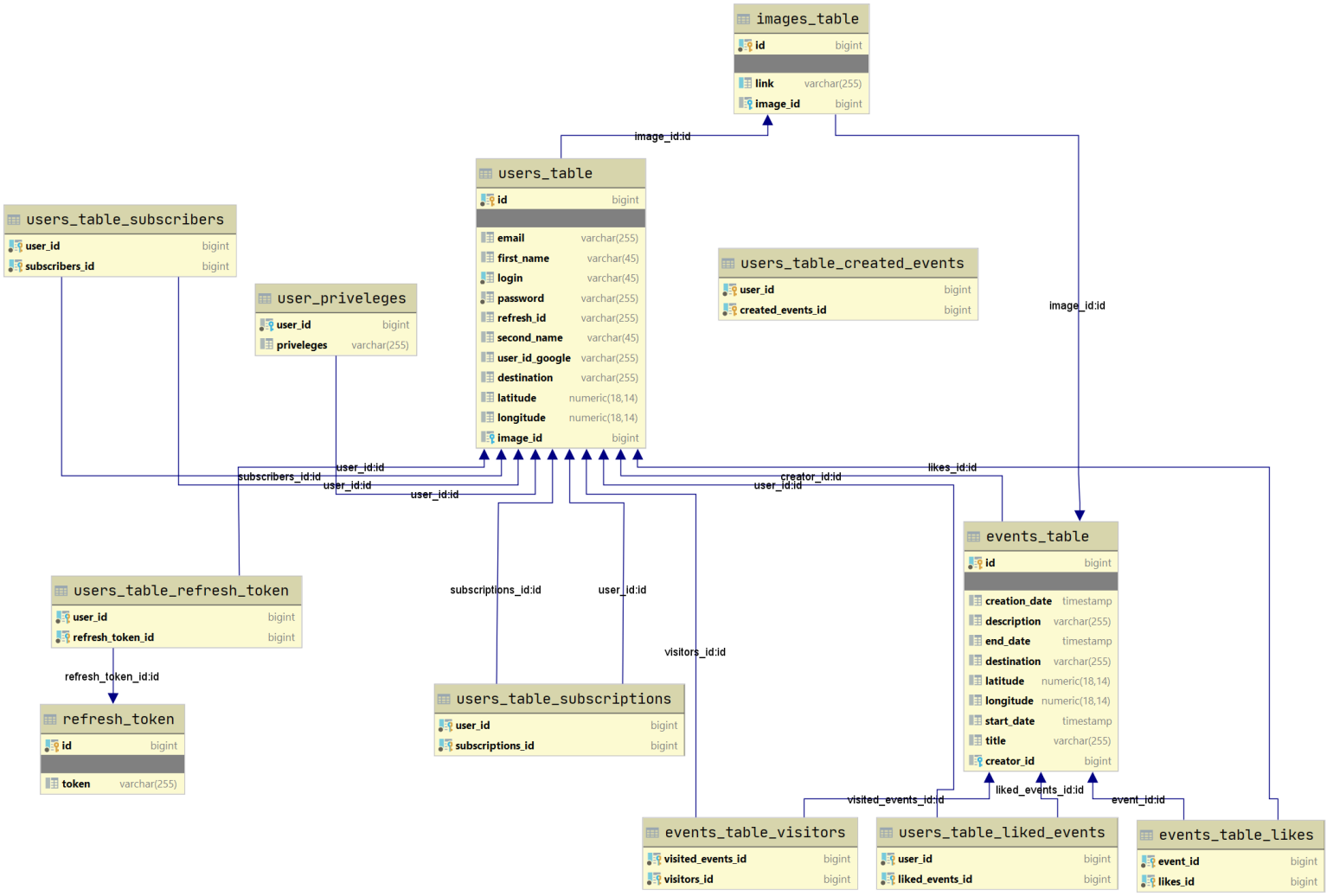
```
alter table refresh_token owner to postgres;

create table users_table
(
    id bigserial not null
constraint users_table_pkey
primary key,
    email varchar(255),
first_name varchar(45),
    login varchar(45) not null
constraint uk_olw5sfua59bu8teuekjg57iqm
unique,
password varchar(255) not null,
    refresh_id varchar(255),
second_name varchar(45),
    user_id_google varchar(255),
destination varchar(255),
    latitude numeric(18,14),
longitude numeric(18,14),
    image_id bigint
constraint fkdfl70mf6ioh9v261m6s628ga1
references images_table
);

alter table users_table owner to postgres;

alter table events_table
add constraint fk9he5aae9p0bn011ucchweplgn
foreign key (creator_id) references users_table;

create table events_table_likes
(
    event_id bigint not null
constraint fk403ilui29ifyylfey49c7nat
references events_table,
    likes_id bigint not null
constraint fkdx9e9ig7jw3vdfm36kx2b2idk
references users_table,
    constraint events_table_likes_pkey
primary key (event_id, likes_id)
);

alter table events_table_likes owner to postgres;
```

```
create table events_table_visitors
(
    visited_events_id bigint not null
constraint fkis03gw5sohgus20e8ldbs5tjb
references events_table,
    visitors_id bigint not null
constraint fkq1wnq7m7euklgs1ygsfo3trqd
references users_table,
    constraint events_table_visitors_pkey
primary key (visited_events_id, visitors_id)
);

alter table events_table_visitors owner to postgres;

create table user_priveleges
(
    user_id bigint not null
constraint fk7xdyub7gw31mf5yge5objb67y
references users_table,
    priveleges varchar(255)
);

alter table user_priveleges owner to postgres;

create table users_table_liked_events
(
    user_id bigint not null
constraint fkboxeq4hlc7d1yb614wrev6auu
references users_table,
    liked_events_id bigint not null
constraint fkcyh0b0k0rqqp2jat6ejgi7ce3
references events_table,
    constraint users_table_liked_events_pkey
primary key (user_id, liked_events_id)
);

alter table users_table_liked_events owner to postgres;

create table users_table_refresh_token
(
    user_id bigint not null
constraint fkh5j5b7ein2650xhnmssjpjwqh
references users_table,
    refresh_token_id bigint not null
constraint uk_ok60wes4njjcj7kwlay14jgsk
```

```
unique
constraint fk8sao2vtk9fungwpxa3ew08fif
references refresh_token,
        constraint users_table_refresh_token_pkey
primary key (user_id, refresh_token_id)
);

alter table users_table_refresh_token owner to postgres;

create table users_table_subscribers
(
        user_id bigint not null
constraint fknvrvsq4pk4irvbqp0d2m2s89i
references users_table,
        subscribers_id bigint not null
constraint uk_rrbeqcl8oxt873nppveomtqp8
unique
constraint fkfudgifsru6unxyetdm9w2uvt4
references users_table,
        constraint users_table_subscribers_pkey
primary key (user_id, subscribers_id)
);

alter table users_table_subscribers owner to postgres;

create table users_table_subscriptions
(
        user_id bigint not null
constraint fkn008tmafstqfteb36c7ni26lw
references users_table,
        subscriptions_id bigint not null
constraint uk_bny5blpv9pe27u1dmoxdfyygp
unique
constraint fk9clh5vm7mcjfwmrxkddfxiput
references users_table,
        constraint users_table_subscriptions_pkey
primary key (user_id, subscriptions_id)
);

alter table users_table_subscriptions owner to postgres;
```



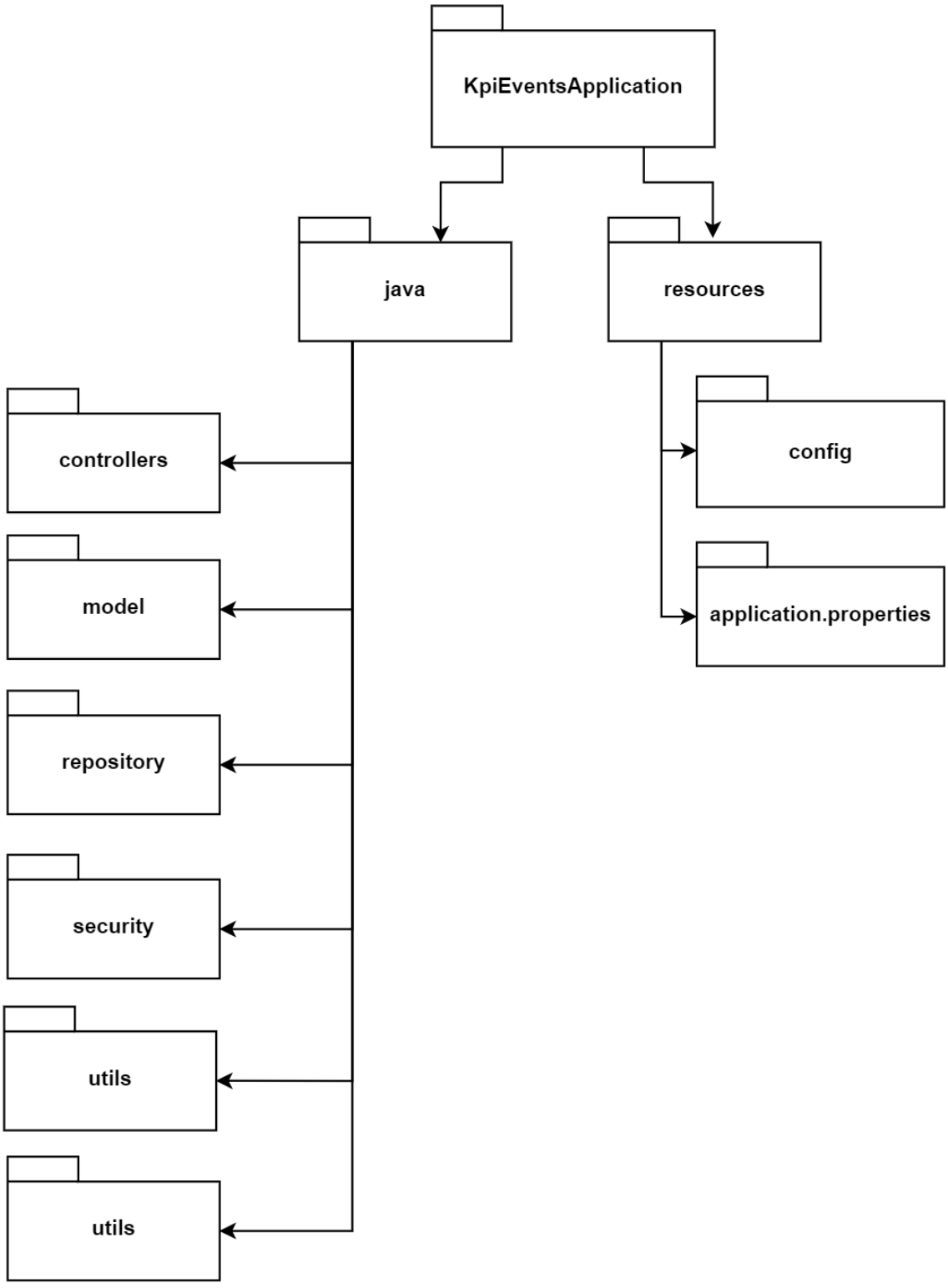
					ДП 6402. 00.004 Д1	
Зм.	Арк	№ докум.	Підпис	Дата		
		Вінницький В.А.			Функціональна схема.	Літ. / Аркуш: 1 / Аркушів: 1
Перевірив.		Каплунов А.В.				
.						НТУУ "КПІ ім. Ігоря Сікорського" ФІОТ гр. ІП-64
Н. Контр.		Сімоненко В.П.				
Затвердив.						



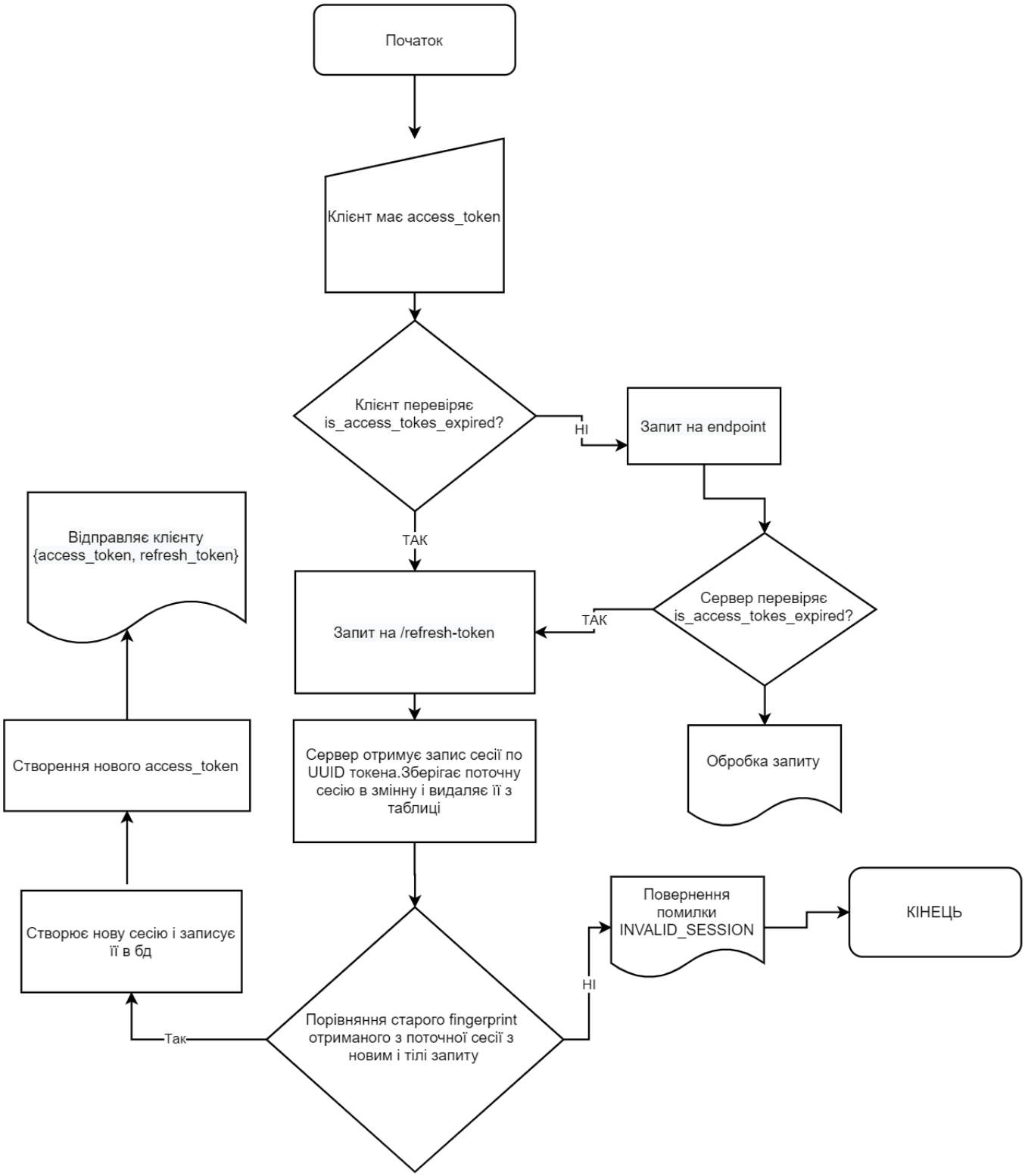
					ДП 6402. 00.005 Д2
Зм.	Арк	№ докум.	Підпис	Дата	
		Вінницький В.А.			Схема бази даних.
Перевірив.		Каплунов А.В..			
.					
Н. Контр.		Сімоненко В.П			
Затвердив.					

Літ.	Аркуш	Аркушів
	1	1

НТУУ "КПІ ім. Ігоря Сікорського" ФІОТ гр. ІП-64



					ДП 6402. 00.006 ДЗ	
Зм.	*Арк*	*№ докум.*	*Підпис*	*Дата*		
		Вінницький В.А.			*Схема пакетів системи.*	*Літ.* / *Аркуш* / *Аркушів*
Перевірив.		*Каплунов А.В..*				1 / 1
.						*НТУУ "КПІ ім. Ігоря Сікорського" ФІОТ гр. ІП-64*
Н. Контр.		*Сімоненко В.П*				
Затвердив.						



					ДП 6402. 00.007 Д4		
Зм.	*Арк*	*№ докум.*	*Підпис*	*Дата*			
		Вінницький В.А.			*Принципова схема системи.*	*Літ.*	*Аркуш*	*Аркушів*
Перевірив.		*Каплунов А.В..*					1	1
.						*НТУУ "КПІ ім. Ігоря Сікорського" ФІОТ гр. ІП-64*		
Н. Контр.		*Сімоненко В.П*						
Затвердив.								